

**Automatic Organization of Digital Music Documents –  
Sheet Music and Audio**

**Dissertation**

zur

Erlangung des Doktorgrades (Dr. rer. nat.)

der

Mathematisch-Naturwissenschaftlichen Fakultät

der

Rheinischen Friedrich-Wilhelms-Universität Bonn

vorgelegt von

Christian Fremerey

aus

Bonn

Bonn, Mai 2010

Angefertigt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der  
Rheinischen Friedrich-Wilhelms-Universität Bonn

1. Gutachter: Prof. Dr. Michael Clausen  
2. Gutachter: PD Dr. Meinard Müller  
Tag der Promotion: 21. Juli 2010  
Erscheinungsjahr: 2010

# Automatic Organization of Digital Music Documents – Sheet Music and Audio

Christian Fremerey

## Abstract

This thesis presents work towards automatic organization and synchronization of scanned sheet music and digitized audio recordings in the scenario of a digital music library. The organization targeted in the project of this thesis includes the segmentation of sheet music books and audio CD collections into individual songs or movements, mapping the resulting segments to corresponding records in a database of metadata, and temporally synchronizing the sheet music documents and audio recordings that belong to the same piece of music on a bar-wise level. Building up a digital music library with a large collection of digitized sheet music and audio recordings requires automated methods for organizing the data, because a manual organization is too expensive and time-consuming. In this thesis, a complete workflow addressing the practical issues that arise in building up and maintaining a digital music library for synchronized sheet music and audio recordings is presented. Algorithms and approaches for the automatic organization of music documents are proposed and evaluated. We introduce a software application to be used by library employees for the import of new data and editing of existing records that integrates the proposed automatic methods. This application, furthermore, allows semi-automatic or manual interaction where automatic approaches are not yet reliable enough to meet the high quality standards that are expected in a library environment. A prototypical user interface for users of the digital library is presented that allows for applications making explicit use of the synchronization between sheet music books and audio CD collections.

**Keywords:** digital music libraries, sheet music, audio recordings, music synchronization, music alignment, optical music recognition, score-performance matching, user interfaces, automatic document organization, digital music representations, partial synchronization, structural differences



# Acknowledgements

This work would not have been possible without the support, advice, and encouragement I received from the people close to me during the last couple of years. In particular, I would like to express my thanks and gratitude to

- my supervisors Michael Clausen and Meinard Müller who always supported and guided me in exactly the way I needed and who probably taught me more about properly doing science than I even realize today,
- my parents for their never-ending love and extraordinary support,
- my wife for her love, her patience, and for believing in me, as well as her parents who welcomed me in their family and trusted in me like their own child  
我非常感谢你们对我这么好 给我的信心 热情地欢迎我成为家庭的一部分 ,
- my former colleagues at the University of Bonn Frank Kurth, Rolf Bardeli, David Damm, Sebastian Ewert, and Verena Thomas, with whom I had the joyful experience of working together for many years,
- Axel Mosig and his group at PICB in Shanghai without whose unconditional help and support during the final phase of this work things would have been so much harder,
- as well as all my family and friends for giving me comfort and happiness.

Shanghai, May 13th 2010  
Christian Fremerey



# Contents

<b>Abstract</b>	<b>i</b>
<b>Acknowledgements</b>	<b>iii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Main Goals of this Thesis . . . . .	2
1.3 Contributions of this Thesis . . . . .	3
1.4 Related Publications by the Author . . . . .	4
1.5 Related Work . . . . .	6
1.5.1 Digital Music Libraries and Collections . . . . .	6
1.5.2 Synchronization, Alignment, and Score Following . . . . .	8
1.5.3 Content-based Comparison of Music Data . . . . .	12
1.5.4 Other Related Fields . . . . .	13
1.6 Structure of this Thesis . . . . .	14
<b>2 Digital Music Representations</b>	<b>15</b>
2.1 The Data Classes Audio, Symbolic, and Sheet Music . . . . .	15
2.2 A Closer Look at the Data Classes . . . . .	17
2.3 SharpEye Music Reader . . . . .	19
<b>3 Bridging the Gap Between Sheet Music and Audio</b>	<b>23</b>
3.1 Sheet Music-Audio Synchronization . . . . .	23
3.2 Dynamic Time Warping . . . . .	29
3.3 Hidden Markov Models . . . . .	31
3.4 Local Similarity and Mid-Level Representations . . . . .	36
3.5 Subsequence Dynamic Time Warping . . . . .	40

<b>4</b>	<b>PROBADO Music Document Organization System</b>	<b>43</b>
4.1	PROBADO Music Project . . . . .	43
4.2	Real-World Challenges . . . . .	45
4.2.1	Track Segmentation and Identification . . . . .	50
4.2.2	Structural Differences . . . . .	53
4.2.3	OMR Quality and Note Events . . . . .	53
4.3	Framework and Workflow . . . . .	55
4.4	Handling Special Cases . . . . .	62
<b>5</b>	<b>From Sheet Music To Note Events</b>	<b>67</b>
5.1	Optical Music Recognition . . . . .	68
5.2	Postprocessing OMR Data Using Simple Heuristics . . . . .	70
5.2.1	Voice Detection . . . . .	71
5.2.2	Key Signature Correction . . . . .	73
5.2.3	Time Signature Correction . . . . .	74
5.2.4	Execution Hint Removal . . . . .	74
5.2.5	Grandstaff Indentation Detection . . . . .	75
5.3	Iterative Approach for Reducing Inconsistency . . . . .	76
5.4	Detecting Repeats and Jumps . . . . .	85
5.5	Estimation of Onset Times and Tempo . . . . .	89
5.6	Deriving Pitches . . . . .	94
5.7	Orchestral Scores . . . . .	96
<b>6</b>	<b>Track Segmentation and Identification</b>	<b>103</b>
6.1	Definition of the Task . . . . .	104
6.2	Text-Based Comparison . . . . .	106
6.3	Content-Based Comparison . . . . .	112
6.3.1	Baseline Experiment . . . . .	114
6.3.2	Real-World Experiment . . . . .	115
6.3.3	Postprocessing Clean-Up . . . . .	117
6.4	Conclusions . . . . .	118
<b>7</b>	<b>Partial Synchronization</b>	<b>119</b>
7.1	Types of Structural Differences . . . . .	120
7.2	Jumps and Blocks . . . . .	120
7.3	Matching Approach . . . . .	124
7.4	Path Degeneration Approach . . . . .	126
7.5	JumpDTW . . . . .	130
7.5.1	Customizable Start Points and End Points . . . . .	133
7.5.2	Optimum Average Cost . . . . .	134
7.5.3	Special States . . . . .	135
7.5.4	Multi-Scale Approach . . . . .	136
7.5.5	Experiments and Results . . . . .	138
7.5.6	Summary of the JumpDTW Approach . . . . .	141



<b>8 Applications</b>	<b>143</b>
8.1 Score Viewer . . . . .	144
8.2 Audio Viewer . . . . .	146
8.3 Cross-Domain Presentation and Navigation . . . . .	147
8.4 Switching Interpretations . . . . .	149
8.5 Cross-Domain Retrieval . . . . .	150
<b>9 Conclusions</b>	<b>153</b>
<b>A Calculations</b>	<b>157</b>
<b>Bibliography</b>	<b>159</b>
<b>Index</b>	<b>171</b>



# Chapter 1

## Introduction

### 1.1 Motivation

Advances in technology during the past decades have led to a continuous growth and improvement of methods for digitization and storage of digital documents. The classic paradigm of libraries that manage collections of physical items such as books or music CDs is more and more shifting towards digital libraries that offer access to digitized copies of their collections through computers inside their reading rooms or to the user's homes through the Internet. Also serving the purpose of long-term preservation, many libraries have started to digitize their collections and to build up infrastructures for the management and access of the digital documents. In combination with modern computer technology for processing digital documents, the availability of large collections of digital documents opens up the possibility for novel ways of organizing and interacting with the library content that have never been possible before.

The PROBADO project<sup>1</sup> (“PROtotypischer Betrieb Allgemeiner DOkumente”, engl.: Prototypical Operation of Common Documents) is a digital library initiative funded by the DFG (German Research Foundation) that aims at realizing this vision of a digital library stated above. The goal of the project is to develop tools and systems as well as to create methods and workflows that allow academic libraries to treat non-textual documents in the same way as textual documents. Focused at the beginning are the domains of music and 3D graphics. In the part of the project that deals with the music domain, the Bavarian State Library<sup>2</sup> (BSB) located at Munich, Germany, creates, collects, and manages digitizations of music documents and corresponding metadata. The Multimedia Signal Processing Group<sup>3</sup> of the Department of Computer Science III at the University of Bonn, Germany, develops algorithms and tools for the automated organization and semantic linking of the data, for content-based retrieval, as well as for the presentation of and interaction with the content. The music part of the PROBADO project poses the scenario and main motivation of the research efforts conducted

---

<sup>1</sup><http://www.probado.de/en/home.do.htm>

<sup>2</sup><http://www.bsb-muenchen.de>

<sup>3</sup><http://www-mmdb.iai.uni-bonn.de>

in this thesis. This thesis constitutes significant work of the Multimedia Signal Processing Group as a contribution to this project.

The digital music documents considered in the project are, at this point, limited to two main types: scans of printed sheet music books, and digitized audio CD collections. The type of music considered is currently limited to classical music. In the early stages of the project, a focus was set on classical and romantic piano sonatas. Throughout the project, the type of music has been widened and now includes complete works of composers such as Beethoven, Mozart, Liszt, Mendelssohn, and Schubert. Further information can be found in several publications related to the project [136, 72, 34, 13, 73].

## 1.2 Main Goals of this Thesis

The main goal pursued in this thesis is to enable the automatic organization of collections of sheet music and audio recordings in the scenario of a digital music library. In this scenario, music collections are given as scans of printed sheet music books and audio CD recordings in high quality. Furthermore, metadata including information about titles, composers, publishers, and artists are acquired. The automatic organization of the music documents pursued in this thesis includes two main tasks. A first task is to group and link all documents and metadata that represent the same pieces of music. Opposed to doing this on a rather coarse level such as operas, concerts, sonatas, and song cycles, here, we target a finer level separating individual songs and movements. A second task is to temporally align the musical content of the different representations of the same pieces of music. We also refer to this task as *synchronization*. In this thesis, we are particularly interested in the bar-wise synchronization of the sheet music documents and the audio recordings. This allows for interesting and novel applications such as automatic highlighting of the playback position in the sheet music while playing back a recorded interpretation, or visually navigating in audio recordings by means of a sheet music representation.

Even though the idea of synchronizing scores with audio recordings is not new, in our project, we employ optical music recognition (OMR) to extract computer-understandable representation of score from given printed sheet music that has been digitized through scanning. The application of this approach in the scenario of a large-scale real-world digital music library brings up several challenges that have not been addressed in previous approaches. One of these challenges is to consistently segment given documents such as sheet music books and audio CD collections into individual movements or songs, and to group and link them with the corresponding metadata. Another challenge is posed by structural differences in the content of different representations of the same pieces of music. Such structural differences can, for example, be caused by different amounts of repeats or verses being performed in different interpretations. A further challenge affecting all the approaches proposed in this thesis are errors and missing information in the optical music recognition results. Opposed to previous and related work, the focus of this thesis lies not on maximizing the temporal accuracy of synchronization techniques. Instead, the focus is set on accomplishing the prerequisites that must be met before a bar-wise synchronization on the given data can be successful in the first

place. Here, the goal is to minimize the manual effort that is required to organize, segment, identify, and synchronize the data contained in large collections of digital music documents.

The work towards these goals leads to a multitude of very interesting and fundamental problems and challenges. Unfortunately, most of these can only be addressed briefly and superficially to avoid moving away too far from the main task and schedule of the project, which is to develop a working prototypical system. To this end, it is more important to identify and address all the emerging challenges with practical solutions than finding the most efficient and elegant solution for only a subset of these challenges. This inevitably leads to this work only scratching the surface of many topics that would deserve further and more in-depth research efforts in the future. In those cases, this work limits itself to identifying and discussing the open challenges followed by simple and practical solutions and a discussion of ideas and directions for future improvements.

### 1.3 Contributions of this Thesis

The main contributions of this thesis are the identification of requirements and problems, novel computational approaches, and the prototypical framework enabling the practical realization of automated content-based organization of sheet music and audio recordings in digital music libraries. This is achieved through a multitude of individual contributions addressing specific problems that arise in this context. Even though the Chapters 2 and 3 might include minor contributions on their own, their main purpose is to make the reader familiar with the involved types of data and the basic technologies and algorithms. The main contributions of this thesis are contained in Chapters 4 to 8.

In Chapter 4, the challenges emerging in the practical realization of content-based organization of large collections of sheet music and audio recordings, which are usually neglected in purely academic work, are revealed and discussed. An overall concept, a user interface, and a workflow for building up and maintaining an organized repository is proposed and implemented. This concept supports the integration of automatic organization techniques in combination with human supervision, controlling, and editing functionalities, which are indispensable in a digital music library targeting high quality and reliability standards. Chapter 5 contributes heuristic strategies and methods for a streamlined extraction of note events from OMR results that encounter the most critical shortcomings of the data for the success of our scenario. This includes several simple heuristic methods for correcting obvious errors, a more general algorithm for iteratively reducing inconsistency based on collections of simple rules, strategies for detecting repeats and jumps based on text and symbols found in the score data, robustly estimating onset times, tempo, and pitches, as well as handling complex orchestral scores including transposing instruments.

In Chapter 6, the task of track segmentation and identification is introduced as an important step in the automatic organization of music documents and is suitably formalized. The challenges of approaches based on comparison of titles and other meta information extracted from the sheet music, audio CD collections, and metadata databases are discussed and a promising general strategy is proposed. Furthermore, experiments towards a content-based approach to

the task are conducted, revealing that content-based approaches and text-based approaches complement each other well and that the idea of combining the two shows great promise for solving the task with good accuracy and robustness. In Chapter 7, several approaches for handling structural differences, which are critical for score-performance synchronization, are proposed and discussed. The novel JumpDTW algorithm is proposed as an extension of the classic dynamic time warping (DTW) algorithm and experiments are conducted proving its effectiveness for handling structural differences in score-performance synchronization. Several extensions to the JumpDTW algorithm are proposed for increasing its efficiency and adding further use for handling special cases that are relevant in practice. Finally, Chapter 8 contributes the conceptual design, realization, and implementation of powerful applications of sheet music-audio synchronization in an intuitive and attractive user interface.

## 1.4 Related Publications by the Author

In the context of this thesis, several publications with participation of the author have been published. Some of these publications already contain parts of the work published in this thesis. In the following, the related publications by the author are listed in chronological order, and their relations to the individual chapters of this thesis are explained.

- [78] F. Kurth, M. Müller, C. Fremerey, Y. Chang, and M. Clausen, “Automated Synchronization of Scanned Sheet Music with Audio Recordings”, in *Proceedings of the 8th International Conference on Music Information Retrieval*, Vienna, Austria, 2007.

In this publication, the idea of computationally synchronizing spatial regions in scanned sheet music to temporal regions in audio recordings on a bar-wise level using optical music recognition results, chroma features, and dynamic time warping is presented for the first time. A first proof-of-concept implementation of an end-user application is based on the SyncPlayer framework [76, 44]. In this thesis, a detailed discussion of sheet music-audio synchronization is given in Chapter 3. The user interface developed in the context of this thesis is presented in Chapter 8.

- [27] D. Damm, C. Fremerey, F. Kurth, M. Müller, and M. Clausen, “Multimodal Presentation and Browsing of Music”, in *Proceedings of the 10th International Conference on Multimodal Interfaces*, Chania, Crete, Greece, 2008.

This is the first publication presenting the novel user interface discussed in Chapter 8 of this thesis, which presents sheet music books to the user as a freely browsable virtual book in 3D and offers advanced functionality such as switching interpretations.

- [73] F. Kurth, D. Damm, C. Fremerey, M. Müller, and M. Clausen, “A Framework for Managing Multimodal Digitized Music Collections”, in *Proceedings of the 12th European Conference on Research and Advanced Technology for Digital Libraries*, Aarhus, Denmark, 2008.

This paper describes a complete framework for managing collections of sheet music and audio recordings. This description includes an outline of the content-based approach to track segmentation and identification that is described in more detail in [48] and

in Section 6.3 of this thesis. It furthermore presents the application of content-based retrieval using the score viewer interface as described in Section 8.5 of this thesis.

- [48] C. Fremerey, M. Müller, F. Kurth, and M. Clausen, “Automatic Mapping of Scanned Sheet Music to Audio Recordings”, in *Proceedings of the 9th International Conference on Music Information Retrieval*, Philadelphia, USA, 2008.

This work presents experiments and results related to content-based comparison of symbolic OMR data and acoustic data extracted from audio recordings targeting the application of track segmentation and identification. These experiments are presented in this thesis in Section 6.3.

- [46] C. Fremerey, M. Müller, and M. Clausen, “Towards Bridging the Gap between Sheet Music and Audio”, in *Knowledge representation for intelligent music processing*, (E. Selfridge-Field, F. Wiering, and G. A. Wiggins, eds.), no. 09051 in Dagstuhl Seminar Proceedings, (Dagstuhl, Germany), Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany, 2009.

This paper contains a discussion of different classes of digital music representations and transformations between them, similar to the one found in Section 2.1 of this thesis. Basic practical difficulties that arise in the task of automizing sheet music-audio synchronization are identified and discussed, which relates to Section 4.2 of this thesis. Aspects and shortcomings of OMR that play a role in Chapter 5 of this thesis are discussed.

- [45] C. Fremerey, M. Clausen, M. Müller, S. Ewert, “Sheet Music-Audio Identification”, in *Proceedings of the 10th International Conference on Music Information Retrieval (ISMIR 2009)*, Kobe, Japan, 2009.

This paper first introduces the concept of bar sequences for the sheet music of a piece of music and for a corresponding performance and possible differences between the two. This concept is used in an extended form in this thesis, see for example Section 4.2. Furthermore, the use of staff signature annotations for handling orchestral music is discussed, as is also used in Section 5.7 of this thesis. Finally, different settings for matching techniques used in the scenario of identifying structural differences between a given score representation and audio representation, as are used in Section 7.3 of this thesis, are evaluated through experiments.

- [141] V. Thomas, C. Fremerey, D. Damm, and M. Clausen, “SLAVE: A Score-Lyrics-Audio-Video-Explorer”, in *Proceedings of the 10th International Conference on Music Information Retrieval (ISMIR 2009)*, Kobe, Japan, 2009.

Besides an extension to the user interface presented in Chapter 8 of this thesis, this paper presents a predecessor of the document organization system proposed in Chapter 4. In contrast to the document organization system proposed in this thesis, this predecessor system is used to manage individual given data sets without being integrated into a larger context and database of a digital music library.

- [93] M. Müller, M. Clausen, V. Konz, S. Ewert, and C. Fremerey “A multimodal way of experiencing and exploring music”, to appear in *Interdisciplinary Science Reviews (ISR), Special Issue: Music and the Sciences, Vol. 35, No. 2*, 2010.

In this article, the synchronization of different types of digital music representations, as found in Chapter 3 of this thesis, as well as the applications of cross-modal presentation and navigation, as described in Chapter 8, are presented as useful aspects in the wider context of experiencing and exploring music.

- [47] C. Fremerey M. Müller, M. Clausen, and “Handling Repeats and Jumps in Score-Performance Synchronization”, to appear in *Proceedings of the 11th International Society for Music Information Retrieval Conference (ISMIR)*, Utrecht, Netherlands, 2010.

This article presents the JumpDTW approach for handling structural differences in sheet music-audio synchronization which is also presented in Section 7.5 of this thesis.

## 1.5 Related Work

Since this thesis, rather than focusing on a specific aspect, covers a multitude of aspects relevant to the general goal of enabling automatic organization of sheet music and audio documents in a digital music library, related work is found in many different areas. In the following, we group the most prominent and closely related fields of work in individual subsections, followed by a final subsection covering several more distantly related fields.

### 1.5.1 Digital Music Libraries and Collections

One goal of the project described in this thesis is to create a digital music library system that handles sheet music and audio recordings. In the following, we briefly introduce projects with similar or related goals. This includes digital music library projects as well as projects for building up collections of digitized music. Some of these projects focus on sheet music or audio recordings only, and the projects vary in terms of access and presentation offered to the user. Some are simply collections of downloadable documents with some basic attached metadata. Others offer specialized systems for access including search for metadata, content-based retrieval, and specialized clients for presentation, navigation, browsing, and annotation. Depending on the complexity, sophisticated document organization systems are needed to make new data available through the system. To our knowledge, no project has reported on successfully performing bar-wise synchronization of score and audio data on a larger scale so far. However, having both types of data available, it is an obvious goal to pursue, since it offers many interesting and useful applications. Therefore, it is likely that some of these projects have already started working on this task, or will try to incorporate it at some point in the future.

A very prominent digital music library project is the Variations2 project based at Indiana University [40]. Despite many other projects, including the one described in this thesis, the Variations2 system is already in use at Indiana University with content provided by its local Cook Music Library. In the follow-up project Variations3, efforts are being made towards installing and using the system at several other locations and institutions. The system includes both sheet music documents and audio recordings, and the user interface supports synchronization between the two. However, the synchronization data has to be



created manually. Work towards automatic synchronization has been conducted, but has not yet been integrated into the system [123].

Another project that aims at handling both sheet music and audio recordings is currently being conducted by the Swiss RISM organization<sup>4</sup> in collaboration with the Distributed Digital Music Archives and Libraries (DDMAL) of McGill University in Montreal and is titled “Informationspool: Repertory of 19th century’s swiss composers” [144]. This project is still in an early stage. Development towards a user interface for interacting with sheet music that supports (manually created) synchronisation between different physical items of the same piece of music on the level of musical sections or movements has been reported in [62].

Besides these projects that focus on digital music documents including sheet music and audio, there are several larger-scale digital multimedia library projects that might also contain sheet music and audio recordings, but that have not yet reported any details or results on the combination of these particular types of data. Europeana<sup>5</sup> is a European initiative aiming at building up a unified digital library for archiving, preserving, and presenting the cultural treasures of Europe in the “digital world”. CONTENTUS<sup>6</sup> as part of the THESEUS initiative<sup>7</sup> is a project initiated by the German Federal Ministry of Economy and Technology that focuses on developing technologies, best practices, and business models for the “media library of the future”, which includes the handling and integration of digital music documents in digital libraries. Finally, Quaero<sup>8</sup> is a large-scale project for research and realisation of prototypes for access and usage of multimedia information involving French and German public and private research organisations such as the Institut de Recherche et Coordination Acoustique/Musique (IRCAM)<sup>9</sup> in Paris and RWTH Aachen University<sup>10</sup> in Germany.

Several other projects focus on either audio documents or sheet music documents and scores alone. EASAIER<sup>11</sup> (Enabling Access to Sound Archives through Integration, Enrichment, and Retrieval) is a European research project based at the Centre for Digital Music, Queen Mary University of London, UK, creating a framework targeted at digital sound archives [79]. In the project goals<sup>12</sup>, “multi-media synchronisation” is mentioned, but so far no details on this topic have been reported in the publications. Many conventional libraries and collections have started digitizing their content including sheet music and audio and making it accessible through the Internet as far as possible with respect to copyright restrictions. There are certainly too many to mention all of them, so we will limit ourselves to some of them who have been mentioned in the literature related to Music Information Retrieval (MIR). The Lester S. Levy Collection of Sheet Music has digitized their content including over 29,000 pieces of popular American music spanning the years 1780 to 1960 [63]. The collection is based at the Milton S. Eisenhower Library of the John Hopkins University, Baltimore, USA, and is accessible to the public through the Internet<sup>13</sup>. In the context of the digitization project,

---

<sup>4</sup><http://www.rism-ch.org>

<sup>5</sup><http://europeana.eu>

<sup>6</sup><http://www.theseus-programm.de/en-us/theseus-application-scenarios/contentus/default.aspx>

<sup>7</sup><http://www.theseus-programm.de>

<sup>8</sup><http://www.quaero.org>

<sup>9</sup><http://www.ircam.fr>

<sup>10</sup><http://www.rwth-aachen.de>

<sup>11</sup><http://www.easaier.org>

<sup>12</sup><http://www.elec.qmul.ac.uk/easaier/index-1.html>

<sup>13</sup><http://levysheetmusic.mse.jhu.edu>

work has been done towards using OMR to make the content of the sheet music accessible and searchable [84]. The International Music Score Library Project (IMSLP)<sup>14</sup> offers free access to a large collections of digitized sheet music that is free of copyright or that provided by the rightholders free of charge. As of April 2010, the collection contained 58,907 scores representing 23,907 works by a total of 3,167 composers spanning all time periods from ancient to modern. The Swiss Repertoire International des Sources Musicales (RISM)<sup>15</sup> has built up a collection of more than 60,000 printed scores and manuscripts from Swiss libraries and archives. Scores in computer-parsable symbolic formats have been collected at the Stanford University, California, USA [129, 65], which are especially useful in combination with the Humdrum Toolkit for musicological analysis tasks [67]. A community-driven collection of freely available scores in the score typesetting language Lilypond<sup>16</sup> [104] is maintained by the Mutopia Project<sup>17</sup>.

Other prominent public and academic resources for sheet music accessible through the Internet include the Munich Digitisation Centre<sup>18</sup> of the Bavarian State Library in Munich, the Digital Image Archive of Medieval Music (DIAMM)<sup>19</sup> at the University of Oxford, UK, the Chopin Early Editions at the University of Chicago<sup>20</sup>, the Neue Mozart Ausgabe<sup>21</sup> at the Mozarteum Foundation in Austria, and the Sibley Music Library<sup>22</sup> at the Eastman School of Music, University of Rochester. Even though many large collections of audio recordings have been created by both companies as well as public institutions, most of the audio material is still restricted by copyright and, therefore, not freely accessible for public or academic use.

In contrast to the project of this thesis, the music collections referred to above behave more like traditional libraries, where individual items such as sheet music books or audio recordings are catalogued and can be accessed in a digital form. However, only in rare cases different representations of the same piece of music are grouped and linked together, and none of the collections offers temporal synchronization of the musical content on the level of bars at the present time.

### 1.5.2 Synchronization, Alignment, and Score Following

One of the main goals pursued in this thesis is the successful temporal alignment, also referred to as synchronization, of score data obtained from sheet music scans and performances given in form of audio recordings. The task of temporal alignment of digital music representations has been the topic of research for many years. Approaches to this task can be grouped by the types of digital music representations used as input as well as the imposed runtime scenario. We distinguish several types of digital music representation. A more detailed discussion of this topic is given in Chapter 2. At this point, we only distinguish two main types. The

---

<sup>14</sup><http://imslp.org>

<sup>15</sup><http://www.rism-ch.org>

<sup>16</sup><http://lilypond.org>

<sup>17</sup><http://www.mutopiaproject.org>

<sup>18</sup><http://www.digital-collections.de/index.html?c=startseite&l=en>

<sup>19</sup><http://www.diamm.ac.uk>

<sup>20</sup><http://chopin.lib.uchicago.edu>

<sup>21</sup><http://dme.mozarteum.at>

<sup>22</sup><http://www.esm.rochester.edu/sibley>

type referred to as *audio* represents acoustic data encoding the variations in air pressure over time captured by a microphone. Audio data is typically found in audio recordings such as CDs or MP3 files. The second type is referred to as *symbolic* and stands for representations that encode pieces of music through symbols that express musical entities. These music entities can, for example, be note events characterized by an onset time, a duration, and a pitch, as commonly found in MIDI<sup>23</sup> files. They can also be symbols of a musical score, such as notes, rests, and clefs, as commonly found in computer-parsable score formats such as MusicXML<sup>24</sup>, Lilypond<sup>25</sup>, or Humdrum<sup>26</sup>. Temporal alignment of digital music documents always takes two music documents as input. If both documents are of the type *audio*, we speak of *audio-audio* synchronization. If both documents are symbolic, we speak of *symbolic-symbolic* synchronization. Finally, if one representation is symbolic, and the other is of the type *audio*, we speak of *symbolic-audio* synchronization. In this case, we also speak of *cross-domain* synchronization, because the two representations that are to be synchronized “live” in different domains (acoustic domain vs. symbolic domain).

The most interesting case for this thesis is the cross-domain symbolic-audio synchronization, because it includes the cases of aligning musical scores to audio recordings. We also refer to this case as *score-audio* synchronization. For an overview, we refer to [30]. Here we distinguish two runtime scenarios. In the *on-line* scenario, a score is synchronized to audio data in real-time. The target scenario, here, typically is real-time accompaniment of a performing soloist by a computer. In this case, the performance of the soloist is captured by a microphone, based on which the computer has to determine the current position in the known score of the piece of music and playback the accompaniment accordingly. In the *off-line* scenario, the complete score and audio data are already available when the computation is started and the computation is not required to run in real-time. This is, for example, the case in our scenario of automatic document organization for a digital music library where recorded audio performances are aligned to corresponding sheet music. In contrast to the on-line case, to determine the position in the score corresponding to a given position in the audio performance, information from future time positions of the performance can be used. Furthermore, the calculation of the synchronization is not time-critical. This makes this scenario somewhat easier compared to the on-line case and allows for approaches with higher robustness and accuracy. A discussion of advantages and disadvantages of off-line and on-line approaches is found in [125].

With its very appealing application of *automatic computer accompaniment*, which is often also referred to as *score following* or *score-performance matching*, the task of on-line symbolic-audio synchronization has received a lot of attentions from researchers with first systems having been presented in 1984 [29, 143]. A nice description of the task and the presentation of the two early approaches can be found in [124]. Since that time, efforts towards improving automatic computer accompaniment have been made continuously up to the present day [114, 31, 60, 61, 119, 19, 64, 108, 105, 109, 122, 90]. The majority of the approaches use hidden Markov models (HMM) in combination with some form of training and a real-time decoding algorithm to perform the given task. The input performance is usually expected to be mono-

---

<sup>23</sup><http://www.midi.org>

<sup>24</sup><http://www.recordare.com/xml.html>

<sup>25</sup><http://lilypond.org>

<sup>26</sup><http://humdrum.ccarh.org>

phonic. There are two notable state-of-the-art systems that are developed in the community. The “Music Plus One” system developed by Christopher Raphael [119, 120, 121, 122] uses a trained hidden Markov model to align the performance of a soloist to the corresponding score part in a component called “listen”. In a second component called “play”, a Bayesian belief network is used to combine the output of the “listen” component with knowledge of the complete score as well as possible rehearsals with the soloist to get an optimum estimate of the onset time of the next musical event in the score. Instead of synthesizing an accompaniment from symbolic data, audio recordings of the accompaniment part can be used by applying time-stretching in real-time. Furthermore, polyphonic performance input is being incorporated into the system. The development of score following systems at the IRCAM Paris has led to the creation of “Antescofo”, which is currently developed by Arshia Cont [109, 131, 24, 25, 26]. The system is based on a coupled pair of agents called “audio” and “tempo”. The “audio” agent estimates physical times of score events in the incoming audio data, while the “tempo” agent continuously predicts the tempo based on both the score and the audio. In the context of this project, the basic idea of a score following system is extended to incorporate more general aspects of time in music compositions and live interactions in music performances. The system has been used in live concerts and features several compositions specifically created for this system.

Opposed to on-line approaches, off-line approaches to symbolic-audio synchronization promise a higher achievable accuracy. Some possible applications are the multimodal presentation and navigation of scores and audio recordings, see Chapter 8, making use of the aligned symbolic data as ground truth for the audio transcription task [142], or using the aligned symbolic data as a reference for performance analysis [33]. The approaches described in the literature usually use either HMMs as in [123, 125, 107] or dynamic time warping (DTW) as in [110, 66, 28, 135, 1, 99, 32, 43, 103] to calculate the synchronization, both of which are based on dynamic programming. Note that most of the on-line score following approaches can also work off-line with only slight modification. More information about dynamic programming techniques and HMMs for aligning sequences can be found in the literature of biological sequence analysis [41]. Both approaches are furthermore introduced in Chapter 3 of this thesis. In the HMM approaches, the local comparison between small sections of the score data and small sections of the audio data is usually achieved through learning emission probabilities of the HMM states by means of training data and machine learning techniques. In the DTW approaches, several different methods have been used for the local comparison. One method is to use transcription techniques to convert the audio data to symbolic data and then perform a comparison in the symbolic domain as, for example, done in [1, 99]. A second method is to use a direct cross-domain similarity measure between sections of the audio signal and expected pitches in the score [135]. A third method is to convert both data types to a common mid-level representation and doing the comparison by means of a similarity measure within this mid-level domain [66, 142, 43]. The use of mid-level representations for sheet music-audio synchronization is discussed in this thesis in Section 3.4.

For the application of using a symbolic representation aligned to a corresponding audio recording as a reference for measuring temporal aspects of the audio performance, the alignment is required to have a very high temporal resolution and accuracy. Some recent work specifically addresses this task [33, 32, 43, 102]. The computational complexity and memory requirements of the off-line synchronization approaches is usually quadratic in the length of the

piece of music. Therefore, modifications have been proposed to reduce these requirements to linear or almost linear complexity [100, 128, 36]. Even though the accuracy and efficiency of score-audio synchronization are important aspects in general, in this thesis, these aspects only play a secondary role. The main focus of this thesis is set towards solving difficulties in achieving reasonably robust score-audio synchronization that arise in the practical real-world scenario of a large-scale digital music library. These difficulties are discussed in Chapter 4. In the previous approaches to music synchronization discussed above, this challenge has not yet been addressed.

At this point, we want to emphasize the difference between *sheet music-audio synchronization*, as is pursued in this thesis, and various other types of symbolic-audio synchronization that have been described in the literature and are often referred to as score-audio alignment or MIDI-audio alignment. In virtually all of the score-audio scenarios and MIDI-audio scenarios, what is actually performed is a comparison between audio data and note events specified by physical (minutes and seconds) onset times, durations, and pitches. However, note events can be derived or given in many different ways including significantly different types and amounts of information. The way that is closest to an audio recording is MIDI data from a manually created MIDI transcription of the same piece of music that uses roughly the same tempo as the audio performance, includes the correct instrumentation, dynamics and accentuations, and temporal details such as ritardandi and fermatas, and perfectly matches the musical structure of the audio performance. Such data is, for example, created by professional musicians to create instrumental versions of popular songs for use in Karaoke applications [142]. Furthermore, pairs of audio recordings and corresponding MIDI versions of this type are available in the RWC Music Database<sup>27</sup> [58] often used in the community. Slightly further away from an audio performance is MIDI data that is generated from a symbolically encoded score and, therefore, contains no dynamics, agogics or temporal nuances other than those written in the score. Such kind of data is often used for testing approaches to MIDI-audio synchronization, and it is usually assumed that at least the rough global tempo is known and that the musical structure of the MIDI data matches the audio performance. If, instead of MIDI data, only a symbolically encoded musical score is given, the note events must first be derived from the score. Depending on how detailed information is contained in the score data, the resulting note events may already differ more significantly from the audio performance, especially if the score does have no information on the global tempo. The hardest case is that of sheet-music audio synchronization, in which only sheet music scans are given. In this scenario, a symbolically encoded score has to be derived using optical music recognition (OMR). Usually, this score data output by OMR software contains errors such as missing notes or accidentals and does not contain information about the tempo or the execution of repeats and jumps. Clearly, the further away the given or derived note events are from the audio performance, the harder the task of aligning the two gets. The fact that, in this thesis, we face the scenario of aligning sheet music scans to audio recordings, poses a main challenge and a major difference to previous work on score-audio synchronization described in the literature.

When comparing and synchronizing scores and performances, it may happen that one encounters differences in their global musical structure like repeats of musical sections or jumps made in the performance different from those suggested in the score. Such structural differences must be resolved for a successful synchronization, which can be a highly non-trivial

---

<sup>27</sup><http://staff.aist.go.jp/m.goto/RWC-MDB>

task. The task of synchronization in the presence of such structural differences is also referred to as *partial synchronization*. The existence and relevance of this issue is acknowledged, for example, in the work by Raphael [123] and Dixon [36] and has been encountered in several works focusing on the scenario of on-line score following such as [80, 111, 140, 2]. In the work by Pardo [111], which addresses possible differences in musical structure between a given score and a monophonic audio interpretation in score following for jazz music, a HMM is used to allow jumps between the known boundaries of musical sections, similar to the motivation for the approach presented in Section 7.5 of this thesis. In the work by Arzt et al. [2] structural differences are addressed by branching three hypotheses after the end of each (known) musical section. The first hypothesis assumes that the performance continues with the subsequent musical section, the second hypothesis assumes that the performance repeats the current musical section, and the third hypothesis assumes that the subsequent musical section is skipped. After enough time has past in the performance to make a reasonably informed decision, only the most likely hypothesis is kept and followed. In previous work on off-line score-audio synchronization the approach usually taken is to assume that the musical structures of the given music documents are identical. A more general and much harder case of partial synchronization has been approached by Müller [92, 94]. This work addresses the case of having given two music documents that do not easily reveal structural information about musical sections, like, for example, pairs of audio recordings. In Chapter 7 of this thesis, partial synchronization is approached for the sheet music-audio case. Because sheet music data contains structural information that reveals possible candidates for musical sections, automatic computational solutions to this task can deliver more accurate results than in the general case.

Besides for symbolic-audio synchronization, music alignment techniques have been proposed for other combinations of input data types, as, for example, for aligning audio recordings of different interpretations of the same piece of music [36], aligning MIDI performances to symbolically encoded lead sheets [112], or aligning pairs of monophonic folk song melodies to reveal pairs that are related to each other [52]. Another type of symbolic data that can be aligned to audio recordings is lyrics given in form of paragraphs of plain text. A successful alignment could, for example, be used to display the lyrics during the audio playback, or to search and navigate to sections of interest by textual retrieval in the lyrics data as, for example, proposed in [98]. The approaches to lyrics-audio synchronization proposed in the literature usually target popular music [145, 49, 22, 81], and involve steps such as finding and isolating vocal sections and then performing a synchronization based on phone models for the singing voice. For the synchronization step, again, HMMs are the most popular choice. Differences in structure between the lyrics given as paragraphs in a text file and the lyrics as performed in an audio recording are often approached by identifying prominent musical sections such as chorus and verse in the songs and matching them to the paragraphs of the lyrics.

### 1.5.3 Content-based Comparison of Music Data

Despite targeting the alignment of pairs of music documents, content-based comparison of music data is used for many different applications. Most related to this work is the application of *content-based retrieval*. For an overview, see [16, 20]. In content-based retrieval, a section of music data, referred to as *query*, is used to search for similar sections, referred to as *matches*,

in a music database. The data type of the database can be the same as the query or can be different. Especially interesting for this work is the cross-domain case of comparing symbolic data to audio data. Queries and matches can be either complete pieces of music, or sections therein. In case that both queries and matches are complete pieces of music, actually, global alignment techniques as discussed in the previous section can be used to calculate a similarity between the query and each piece of music in the database, as, for example, done in [113, 66]. More challenging is the case where queries and matches can be arbitrary sections of pieces of music. In contrast to only requiring a global comparison between two pieces of music, this case requires mechanisms for finding subsequences with high similarity to the query [77, 138, 137]. Further work has been done for the case of using audio queries to search in databases of audio recordings [97, 91, 89, 106, 77, 74]. Examples for applications other than content-based retrieval that involve content-based comparison of music data are *cover song identification* [133] and the comparison of performance aspects in audio recordings [130].

#### 1.5.4 Other Related Fields

Structural differences between scores and performances play an important role in the work presented in this thesis. A common task that is related to identifying the structural differences between two representations of the same piece of music is the structural analysis of a single representation. Using the results of a successful structure analysis, the structural differences between two given representations of a piece of music could be more easily revealed and resolved. In most works, the task of structural analysis is approached for audio recordings of popular music. Depending on if the approach targets at segmenting the recording into musical sections, or just at identifying chorus sections, one speaks of *audio structure analysis* or *audio thumbnailing* [7, 56, 57, 85, 8, 21, 82]. The approach usually taken is to search for pairs of sections in the audio recording that are similar to each other. Repeating sections are, then, usually identified as chorus or verse sections. The task of audio structure analysis is especially hard for classical music, because the musical structures are often more complex than in popular music. Furthermore, in classical music, similar sections may comprise significant musical variations in tempo, dynamics, and instrumentation [96].

The processing of score data obtained from sheet music scans, as discussed in this thesis, strongly depends on the quality of the optical music recognition (OMR) results. Improvements of OMR would directly benefit the approaches presented in this project. An overview to OMR can be found in [4] and references to older work is found in [9]. Even though OMR has been a active research topic [23, 5, 126, 87, 10, 139, 71, 115], the high variability in music notation, possible complexity of musical scores, and image quality make OMR a very difficult task to solve with high accuracy and robustness. Therefore, OMR systems usually assume that users correct recognition errors manually. Adaptive approaches to OMR make use of these manual corrections to adapt to the given type of score and image quality to achieve improved accuracy over the process of recognizing larger collections of digitized sheet music [50, 38, 84, 37, 116, 117].

The task of deriving a score or other symbolic representation of a piece of music from an audio recording is called *music transcription*, see for example [70, 12, 146]. Using music transcription, one can convert audio recordings to the symbolic domain and then perform

the synchronization to a given score in the symbolic domain. However, music transcription, in general, is very hard, or maybe even ill-posed for some types of music. In particular, it involves making hard decision on the existence and exact parameters of note events based on the audio data only, which is much harder and much more error-prone than the comparison of the audio data and the given score. Therefore, incorporating this very hard and error-prone step in an approach to solve an actually simpler task (comparison of score and audio data) seems not to be an optimum choice.

An final important related field to the work presented in Chapter 8 of this thesis is the study and design of user interfaces. Here, related work has been presented in the context of user interfaces for digital music libraries that incorporate the display of scanned sheet music and possibly the playback of audio recordings [3, 6, 62].

## 1.6 Structure of this Thesis

The thesis is organized as follows. Chapter 2 presents the types of digital music documents that play a role throughout the thesis. In Chapter 3 the basics of synchronizing sheet music and audio recordings as well as searching a database of audio recordings for segments that are similar to a given score excerpt are explained. In Chapter 4 we make the transition from the “friendly” conditions assumed in Chapter 3 to the real-world scenario of a large-scale digital music library. Here we identify and discuss the problems that arise when pursuing the goal of synchronizing large collections of scanned sheet music to digital audio recordings. Furthermore, a document organization system that can be used by library employees to build up and manage a digital music library is presented. Chapters 5 to 7 focus on solutions to the challenges identified in Chapter 4. Chapter 5 discusses approaches to improving the use of OMR output for our application scenario of automatic document organization. Approaches and experiments towards the segmentation and identification of individual movements or songs from sheet music books and audio collections are presented in Chapter 6. In Chapter 7, we propose novel approaches to compute a synchronization between scores and performances in the presence of structural differences. The techniques discussed in Chapters 5 to 7 can be integrated into the document organization system presented in Chapter 4 to minimize the manual interaction that is needed in building up and maintaining large organized collections of synchronized sheet music books and audio CD collections. In Chapter 8, we present a user interface and several applications that make explicit use of the synchronization between scanned sheet music books and audio CD collections. The thesis closes with conclusions and a discussion of possible future directions in Chapter 9.



## Chapter 2

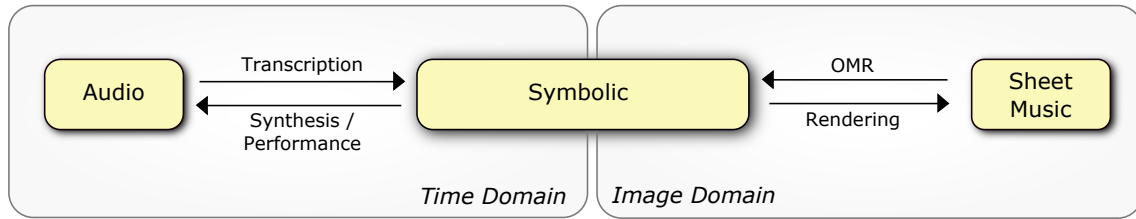
# Digital Music Representations

There are many different types of digital music representations that capture aspects of music on various different levels of abstraction. In this chapter, we discuss digital music representations that are relevant for the synchronization of sheet music and audio recordings of Western classical music. Given a sheet music representation and an audio recording of the same piece of music, the task of synchronization between the two is to find a mapping between two-dimensional regions representing bars in the sheet music and corresponding temporal regions in the audio recording. This task is discussed in more detail in Chapter 3. The goal of this chapter is to give a basic introduction to the different digital representations and transformation between representations of music that are used in the subsequent chapters of this thesis. Section 2.1 introduces three main classes of music representations, namely *Audio*, *Symbolic* and *Sheet Music*. These three classes are further broken down into subclasses in Section 2.2. Finally, in Section 2.3, we introduce the OMR software used in this thesis and describe the type of output it delivers.

### 2.1 The Data Classes Audio, Symbolic, and Sheet Music

We distinguish three main classes of music representations: *Audio*, *Symbolic* and *Sheet Music*, see Figure 2.1. The class *Audio* stands for audio recordings suitable for acoustic reproduction as given in formats such as WAV or MP3. The class *Symbolic* stands for any kind of representation of a piece of music that is based on musically meaningful symbols such as notes or bars. Example formats for symbolic representations are MusicXML, Humdrum, LilyPond and MIDI. Finally, the class *Sheet Music* stands for visual representations of a score as encoded in TIFF, PDF, or other image formats.

In this thesis, we use the term *symbolic* to refer to any data format that explicitly represents musical entities. The musical entities may range from note events with explicit timing information, as can be found in MIDI files [88], to graphical shapes with attached musical meaning as is the case in the SCORE engraving system [134]. Examples for symbols that do not represent musical entities are audio samples in audio files or pixels in bitmap image files. Also the



**Figure 2.1.** Illustration of three main classes of music representations relevant in this thesis: Audio, Symbolic and Sheet Music.

graphical shapes in vector graphics representations of scores are not considered to be musical entities as long as they do not additionally specify the abstract musical entity represented by that shape. Note that audio data is considered to live in the time domain, i.e., positions and regions are measured in units of time. On the contrary, sheet music is considered to live in the image domain, i.e., positions and regions are specified in the two-dimensional Euclidean space. Symbolic score data can contain both time domain and image domain information. Actually, it is in the symbolic realm where the transition between the image domain and time domain is made.

Certainly, there is a wide range of digital music representations that are to be considered symbolic. A discussion of symbolic formats up to the year 1997 can be found in [132]. Since then, however, countless new formats have been proposed and developed. These include both open and well-documented formats as well as proprietary formats that are bound to specific software packages. A quite extensive list of current software applications that deal with symbolic music formats can be found in the context of conversion from and to the MusicXML format developed and maintained by Recordare<sup>1</sup> [55]. Some other examples for symbolic data formats are LilyPond, Humdrum, NIFF, MIDI and SCORE. Example formats for audio data are WAV and MP3. Example formats for sheet music are BMP, TIFF and PDF.

Having specified the meaning of music representation classes, we now identify transformations between these classes. In Figure 2.1, these transformations are indicated by arrows. The transformation from sheet music to symbolic score data is commonly referred to as *optical music recognition* (OMR). This includes, for example, the recognition of musical symbols from scans of printed sheet music pages. The reverse transformation, i.e., the creation of a sheet music image from symbolic score data, is called *rendering*. Depending on how detailed geometric information is given in the symbolic score data, this step may or may not require the application of typesetting and engraving rules. The task of creating symbolic score data from an audio recording is called *transcription*. A transformation in the reverse direction is called *synthesis* or *performance*. Note that these transformations do not always preserve all information and therefore may not be reversible.

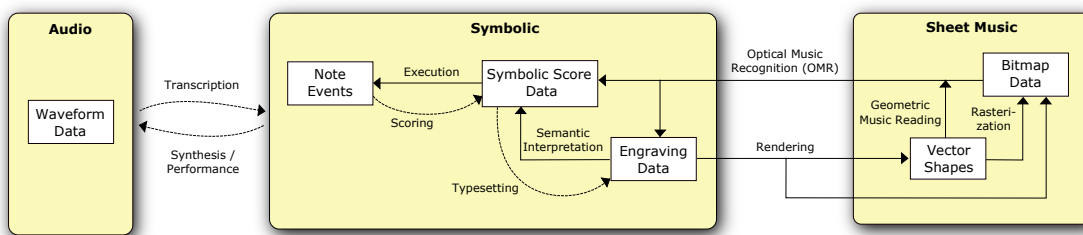
For the scenario of a digital music library, the availability of data plays an important role in the decision of what formats to consider. Even though efforts have been made in building up collections of symbolic score data, as for example in [129, 65] or [83], the vast majority of classical musical scores is still only available in form of printed sheet music. Libraries

<sup>1</sup><http://www.recordare.com/xml.html>, March 2010

have recently started to digitize their sheet music collections through scanning resulting in growing collections of scanned sheet music. Using optical music recognition, these scans become available for analysis and automated processing in digital music libraries.

## 2.2 A Closer Look at the Data Classes

In the previous section, we have specified three main classes of digital music representations. In this section, we will take a closer look at these classes and distinguish several information subclasses within the main classes. These information subclasses allow us to discuss transformations between and within the classes in more detail.



**Figure 2.2.** A closer look at the data classes of Figure 2.1 reveals several information subclasses.

First, we distinguish between two subclasses in the sheet music class, see Figure 2.2. On the far right end in the figure, we have *bitmap data*. These are raster images that consists of a two-dimensional grid of pixels. Each pixel is assigned a certain value for color or shade. No information is given about what musical symbols are there and which pixels belong to which symbol. Somewhat closer to the symbolic but still part of the sheet music class are *vector shapes*. Here, the sheet music image is encoded as a set of vector shapes, where each shape is specified by parameters such as form, position and size. Musical symbols are usually represented by one or more such shapes. However, the type and meaning of the musical symbols is not encoded in the data.

In the symbolic class, the information subclass that is closest to sheet music is *engraving data*. Here, musical symbols are specified explicitly along with information about their shape, position and size. The difference between engraving data and vector shapes is that in engraving data the type of the musical symbol is known, e.g., note head, augmentation dot, staccato dot, bar line or note stem. A bit further away from the sheet music class is the information subclass called *symbolic score data*. This subclass contains explicit information about the musical symbols like clefs, time signatures, notes, rests, accidentals, and dynamics, but does not contain information about the layout and particular shape of the symbols. Another symbolic information subclass that is commonly used in music information retrieval is *note events*. Here, the music is represented as a sequence of note events, where each event has properties such as onset time, pitch, and duration. Such a representation is already part of the temporal domain and, therefore, is closer to the audio class. In the audio class, the only information subclass we consider is the *waveform data*. Here, an audio signal encoding the variations in sound pressure over time is represented by a sequence of samples.

Note that data formats that store digital representations of pieces of music as discussed in Section 2.1 may actually contain information in more than a single information subclass. For example, a single format might store symbolic score data as well as some information related to note events and engraving data. Throughout this thesis, we assume that waveform data has been obtained through acoustical recording of real performances of classical music. Bitmap data is obtained through digitization, i.e., scanning, of printed sheet music. The other types of data are obtained through data transformation, which, in Figure 2.2, are indicated by arrows. The transformation of bitmap data to symbolic score data and engraving data is called *optical music recognition* (OMR). For differentiation, we call the somewhat easier transformation from vector shapes to symbolic score data and engraving data *geometric music reading*. Engraving data can be used to either create vector shapes or bitmap data, both of which we refer to as *rendering*. The process of getting from vector shapes to bitmap data is called *rasterization*. Engraving data can also be used to generate symbolic score data. The transformation that is necessary to do so is referred to as *semantic interpretation*. Having symbolic score data available, it is possible to generate corresponding note events through a transformation we call *execution*.

All the transformations mentioned above are more or less deterministic, which means that no additional information has to be introduced to complete the transformation. Some minor exceptions are the execution of trills and ornaments and the resolution in the rasterization. On the contrary to these somewhat deterministic transformations, there are transformations that have some significant degrees of freedom. In Figure 2.2, these transformations are indicated by curved and dashed arrows. The transformation from note events to symbolic score data may be referred to as *scoring* and requires some decisions, for example, about how to translate onsets, pitches and durations into notes and rests in staves and bars. Another transformation that has some degrees of freedom is called *typesetting*. It refers to the transformation from symbolic score data to engraving data, effectively requiring to make decisions about the exact size, shape, layout and positioning of musical symbols. The transformations that introduce additional information implicate that in the transformation going in the opposite direction some information is lost or discarded. In our case, information is discarded in the transformations *semantic interpretation* and *execution*.

Some of the above transformations are considered easier to achieve than others. A transformation that is considered hard and that plays an important role in this work is optical music recognition (OMR). During this process, musical symbols have to be recognized and interpreted from the mere set of pixels available from the given bitmap data. The task is particularly hard because symbols do not always look exactly the same and may be degraded in quality from artifacts introduced by the printing or scanning process. Furthermore, many symbols overlap with staff lines, and in some cases additional overlaps occur between non-staff-line symbols. Finally, musical scores and the interrelations between musical symbols can become quite complex. Recognizing and correctly interpreting the meaning of all the musical symbols requires the use of high-level semantic rules of music notation, which is easy for humans but hard to implement algorithmically. Typical optical music recognition software usually mainly outputs symbolic score data. Engraving data is output sparsely or not at all, even though the data must be available in the process of getting to the symbolic score data anyway. The OMR process gets a lot easier, when the input data is vector shapes created with a music notation or engraving software (geometric music reading), see [101]. Here, the

symbols or symbol parts are explicitly given and are without any degradation. This makes the recognition and classification of symbols simple and robust, and this poses a big advantage in the stage of deriving symbolic score data.

Note that the transformations between the audio class and symbolic class also involve some degrees of freedom. In the case of synthesis, this is the choice of the synthesizer and the generation of synthesizer controlling data from the given symbolic data. In the case of a performance the degrees of freedom include aspects of the musical interpretation, physical articulation on musical instruments, room acoustics and recording setup. In the transcription task, all of these must be reverted to extract data for a given symbolic data model, like for example note events. In case of bigger musical ensembles involving orchestras and choirs, however, the waveform might not include enough information to uniquely reconstruct which notes have been played at what time by which instrument.

## 2.3 SharpEye Music Reader

In the PROBADO project, which delivers the main motivation for this thesis, optical music recognition is used to obtain symbolic score representations from scanned sheet music. To this end, the commercially available OMR software SharpEye Music Reader<sup>2</sup> 2.68 is used as the main module for the OMR task. The software consists of two main components. The so-called Liszt music-OCR engine is a command-line application that takes bitmap images as an input, performs optical music recognition, and outputs the recognition results in a text-format file using the extension `*.mro`. We will refer to these output files as *MRO files*. The SharpEye 2 application itself is a Windows application providing a graphical user-interface (GUI) as a frontend to the OMR engine. Besides offering functions to open, preview, and apply OMR to input image files, it also has a built-in editor for inspecting and correcting the OMR results. In addition to that, the frontend offers functionalities for exporting to MIDI, MusicXML, and NIFF files. The editor in the frontend does not only read the MRO files, but it also performs some additional semantic interpretation of the data. To be more specific, the editor is capable of estimating concurrent voices in situations where several voices are notated in a single staff. This is, for example, useful for calculating the correct musical onset times of the notes. Even though the GUI application can handle such aspects, they cannot be represented or saved in the current version of the MRO file format.

For the project work described in this thesis, we directly use the Liszt music-OCR engine to perform OMR on scanned score images instead of using the SharpEye 2 GUI application. This approach has several advantages:

- As a command-line application, the OMR engine can easily be automated by calls from our own software components.
- The MRO format is documented<sup>3</sup> and can be parsed by our own software components.

---

<sup>2</sup><http://www.visiv.co.uk>, September 2009

<sup>3</sup><http://www.visiv.co.uk/tech-mro.htm>, Accessed: September 2008

**Figure 2.3.** Score excerpts with highlighted examples of symbols and information that are not output by the OMR software.

- The original MRO files contain more low-level information output by the OMR engine than the files that can be exported from the GUI application. This low-level information allows us to perform our own semantic interpretations and corrections.

In addition to a subjective evaluation of the quality of OMR results, the above listed advantages also make this approach the preferred choice over using other commercially distributed OMR software such as SmartScore<sup>4</sup> or PhotoScore<sup>5</sup>.

To give an impression of the data that is obtained through the OMR process using SharpEye, Table 2.1 lists the main types of musical entities that are output in the MRO files along with a subjective impression of their respective recognition reliability based on the experiences made in the project. In general, one can say that most of the symbols relevant for determining pitches and relative onset times are correctly recognized for the better part of the notes present in the score. Position and layout information of symbols and staves is preserved with good accuracy. Accidentals, key signatures, and beams are quite often missed out. Furthermore title headings, textual tempo directives, and textual jump directives are often missed out or are erroneous to a degree that makes them of little use in automated processing. Misclassified symbols occur only occasionally, because the engine rather tends to omit symbols such as notes, clefs, and accidentals instead of outputting misclassified symbols.

We close this chapter with a brief discussion of symbols and information contained in the sheet music that would be useful for achieving the goals pursued in this thesis, but that are not contained in the OMR results. Some examples are depicted in Figure 2.3. The lack of this information needs to be taken into consideration in the approaches discussed in the remainder of this thesis.

<sup>4</sup><http://www.musitek.com>, September 2009

<sup>5</sup><http://www.neuratron.com/photoscore.htm>, September 2009

- **Note time and absolute pitch:** This information is required for deriving note events from the symbolic score data. The approach taken in this project to derive absolute time and pitch from the OMR results is described in Chapter 5.
- **Alternative endings for repeats:** Usually indicated by brackets above the grand staff (see Figure 2.3), this information is required to derive the correct sequence of bars that is to be performed or synthesized in a realization of the score.
- **Segnos:** These markers usually indicate a jump to a different position in the score.
- **Tempo directives:** These directives usually appear in form of text or a note symbol with a BPM (beats per minute) count. The tempo information is required to convert musical times such as quaters and eights to physical times such as minutes and seconds.
- **Track boundaries:** A set of pages of scanned sheet music can contain more than one piece of music possibly including several songs or movements, which, throughout this thesis, are referred to as *tracks*. The track boundary information says at which positions in the score a track begins or ends.
- **Title headings and section headings:** The title heading can include information on the musical work including name, composer, and possibly a catalog number such as “opus”. It is important for identifying which particular musical work or movement is represented by the given score. A section heading is a title for a section of a piece of music that does not necessarily indicate the beginning of a new track.
- **Jump directives:** Even though repeat signs are recognized, the OMR data does not explicitly say how these translate to jumps from a source bar to a target bar. Textual jump directives, if not omitted, are output as regular text that is not further classified in its function.
- **Transposing instruments:** These are instruments for which the sounding pitch produced by the instrument is different from the pitch that is written in the score. This information is required to determine the correct pitches for deriving note events from the score.
- **Concurrent voices:** The notes in a single staff might be split across several concurrent voices. This information is important for determining onset times for the notes found in the score.

staves & grand staves	usually reliable except for complex orchestral scores with varying number of staves per grand staff and some image degradations
bar lines	reliable except for rare cases where a bar line is interrupted by some symbol or text; bar line types (e.g., single, double, thin-thick) are also quite reliable
repeat signs	reliable except for cases where the repeat signs are surrounded by brackets or parantheses or rare cases where the repeat signs touch a staff line
clefs	fairly reliable; alto and tenor clefs seem to be less reliable than treble and bass clefs; clefs are sometimes omitted, but not misclassified as a wrong clef
key signatures	fairly unreliable; in many cases some of the accidentals of a key signature symbol are omitted resulting in a key signature of the wrong key
time signatures	not very reliable; in piano music, time signatures were sometimes recognized with wrong numbers; in orchestral music, time signatures were sometimes erroneously inserted
note heads & rests	reliable to a high degree; sometimes, notes or rests were omitted, but only a fractional part compared to the correctly detected note heads and rests; erroneously inserted notes or rests occurred only rarely
beams & flags	not very reliable; in more complex passages or in presence of image degradations, beams and flags were often omitted
accidentals	not very reliable; sometimes omitted; in rare cases also misclassified as note head
accentuations	are recognized and supported in the MRO files but are not used in the project so far
slurs	are recognized and supported in the MRO files (approximated as an arc) but are not used in the project so far
grace notes	are recognized, but often erroneously because of missing beams and flags
dynamics symbols	are recognized and supported in the MRO files but are not used in the project so far
text & lyrics	the MRO format distinguishes lyrics from other text but sometimes confuses the two; recognition of lyrics can be fairly good; recognition of other text is fairly unreliable with text elements often being omitted or recognized with a high error rate

**Table 2.1.** Musical entities and their subjective reliability in the recognition results of the SharpEye Music Reader 2.68 software used in the PROBADO project.



## Chapter 3

# Bridging the Gap Between Sheet Music and Audio

After having discussed the relevant data classes for digital music documents in Chapter 2, in this chapter, we talk about how we can bridge the gap between two of these classes, namely sheet music and audio. A sheet music representation and an audio recording of the same piece of music can be temporally linked by a process we call *sheet music-audio synchronization*. We introduce the task of sheet music-audio synchronization in more detail Section 3.1. Here, both representations are first converted to sequences of temporal frames represented by suitable features. From these sequences, an optimum alignment is calculated employing algorithms based on dynamic programming techniques, such as dynamic time warping (DTW), which is described in Section 3.2, and the Viterbi algorithm in combination with hidden Markov models (HMM), which is described in Section 3.3. We discuss the choice of suitable features for the local comparison of frames and introduce the concept of mid-level representations and chroma features in Section 3.4. Besides synchronization, another important task for bridging the gap between sheet music and audio is to find all documents in a database of one datatype and within these documents all sections that correspond to a given query segment of the other datatype. We refer to this task as *sheet music-audio matching*. An algorithmic approach to the matching task is *subsequence dynamic time warping* (SSDTW), which is basically a generalization of the DTW approach. This approach is presented in Section 3.5.

### 3.1 Sheet Music-Audio Synchronization

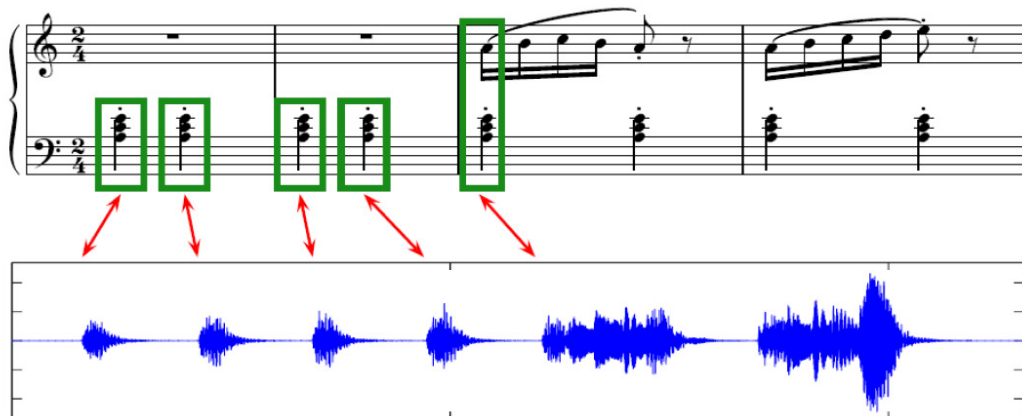
In the previous chapter we have discussed several classes of digital representations for a piece of music. The two most common classes are sheet music and audio. These two classes complement each other regarding level of abstraction and perceptual modality. While sheet music visually presents music on a rather compact symbolic level, audio recordings acoustically present a performance of the music, which may include subtleties and details regarding the musical interpretation and the sound of instruments and room acoustics. For the presentation

and interaction with music, it would be desirable to have both representations available and linked together in a way that for each position in the audio representation the corresponding position in the sheet music representation is known and vice versa. Having such a linked pair of representations opens up many interesting applications, some of which will be discussed in the following chapters.

There are many scenarios of how one can obtain a linked pair of sheet music and audio representation. Looking at Figure 2.2, given any single one of the illustrated representations, one could obtain both a sheet music and an audio representation from it by performing the required transformations. For example, given an audio recording, one could perform transcription, scoring, engraving, and rendering to obtain a sheet music representation. By simply keeping book of the times of the note events in the audio, the onset time of each note in the resulting sheet music representation is known and the sheet music and audio representations can be linked. A second example would be to start with scanned sheet music and then perform OMR, extraction of note events, and synthesis to obtain an audio representation. One could also start with symbolic score data and transform it to both an audio representation as well as a sheet music representation.

If starting with only a single given representation, deriving both a sheet music and an audio representation from that always has the disadvantage of involving transformations that are either error-prone (transcription, OMR) or that require augmenting the data with artistic information and decisions (synthesis, scoring, typesetting). Much more interesting for a multimodal presentation, therefore, is the scenario of starting with both audio recordings of real performances and professionally typeset score editions given in high quality. In this scenario, the scores may be given either as bitmap data, vector shapes, or symbolic score data. In any of these cases, both an image representation and a symbolic representation of the score can be derived without the involvement of transcription, synthesis or scoring. Due to copyright restrictions and commercial interests of publishers, high quality symbolic score editions are hard to get for use in a digital music library. Many sheet music editions, especially older ones that are already public domain, are only available in printed form. In the project described in this thesis, we, therefore, focus on the scenario of having given the scores in form of scans of printed sheet music. In particular, we work with classical Western music spanning instrumentations from piano music to symphonic music including works by composers such as Beethoven, Mozart, Liszt, Mendelssohn and Schubert.

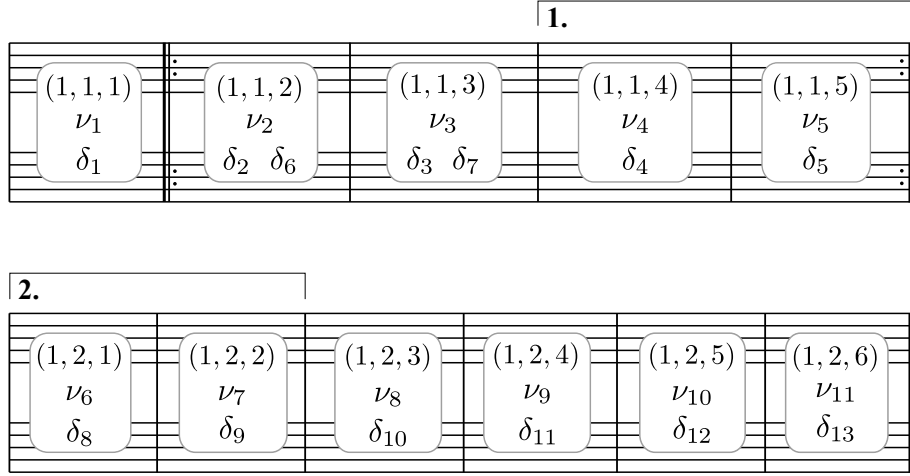
To link the given sheet music and audio data as described above, we need to perform a process we refer to as *sheet music-audio synchronization*. Instead of synchronization, the process is often also called *alignment* in the literature. The goal of sheet music-audio synchronization is to link regions within the two-dimensional image domain of sheet music documents to semantically corresponding temporal regions in audio recordings, see Figure 3.1. One may choose one of several options for the granularity of the regions that are to be linked, for example, pages, lines, bars or notes. Depending on the choice of granularity, there are different strategies to perform the synchronization task. For example, in case of a page-wise synchronization, the manual creation of links between the pages of sheet music and corresponding temporal regions of audio recordings may still be acceptable. However, aiming at finer grained resolutions, the complexity for creating the links increases significantly, hence requiring automated synchronization methods.



**Figure 3.1.** Illustration of the sheet music-audio synchronization by means of the first four measures of Op. 100, No. 2 by Friedrich Burgmüller. The figure shows a scanned musical score and the waveform of a recording of the same measures. The synchronization is indicated by red bidirectional arrows linking regions (given as pixel coordinates) within the scanned image and physical time positions within the audio recording.

In this thesis, we aim at a synchronization on bar-wise level. For the type of music used in this project, bars are a canonical and more or less evenly-spaced segmentation provided by the sheet music. The division into bars is reasonably fine for interesting applications and is coarse enough to provide some robustness against local inaccuracies within bars that might be caused by OMR errors. In the following, we assume that we are given one scanned sheet music representation and one audio interpretation of the same piece of music. The sheet music is organized as a set of bars. We assign a unique label (**page number**, **line number**, **bar number**) to each bar written in the sheet music, where **page number** is the number of the page, **line number** is the line number on the page in top-to-bottom order, and **bar number** is the bar number in the line in left-to-right order, see Figure 3.2. Here, we use the term *line* to refer to what is often called “grand staff” or “system” in musical terms. In this context, the term *bar* refers to any horizontal section of a grand staff that is delimited by either bar lines or the left and right boundaries of the grand staff. Note that this is a strictly technical way of assigning bar labels, and that these labels usually do not coincide with musical bar numbers as they are commonly used in Western sheet music notation. We use this type of labels throughout the whole thesis to refer to bars in the sheet music, and also to specify locations and regions in sheet music books as will be discussed later on.

Let  $\mathcal{B}$  denote the set of all bar labels of the piece. Ordering this set in a canonical way, i.e., ordering by **page**, then **line**, then **bar**, we get the *notation bar sequence*  $\nu = (\nu_1, \dots, \nu_B)$  with  $\nu_b \in \mathcal{B}$ . The length of the notation bar sequence is  $B = |\mathcal{B}|$ . Sheet music may contain jump directives like repeat signs, alternative endings, da capos or segnos. Following these directives as they are written in the sheet music, one obtains a sequence  $\delta = (\delta_1, \dots, \delta_D)$ ,  $\delta_d \in \mathcal{B}$ , indicating the *default bar sequence* that is to be played when performing the piece. Figure 3.2 illustrates the bar labels  $\mathcal{B}$ , the notation bar sequence  $\nu$ , and the default bar sequence  $\delta$  for an example set of bars as it might be found in piano music. Note that, throughout this thesis, we use the term *bar* to refer to different things, depending on the context. For example, we might use the term *bar* to refer to an element of  $\mathcal{B}$ , its corresponding region in the sheet music



**Figure 3.2.** Illustration of bar labels  $\mathcal{B}$ , the notation bar sequence  $\nu$ , and the default bar sequence  $\delta$ .

image that represents the bar, the musical content of the bar, or an occurrence of the bar in a bar sequence such as the notation bar sequence  $\nu$  or the default bar sequence  $\delta$ .

Let  $T_{\text{audio}}$  be the duration of the audio recording in seconds. For now, we assume that the bar sequence played in the audio interpretation exactly matches the default bar sequence  $\delta$  and that there is no additional silence at the beginning or end of the recording. Given the sheet music representation and the audio recording and assuming that the set of bar labels  $\mathcal{B}$  as well as the default bar sequence  $\delta$  have been correctly determined from the sheet music, the task of bar-wise synchronization is to calculate a partition of  $[0, T_{\text{audio}}]$  into time intervals  $I_1, \dots, I_D$  such that time interval  $I_d, d \in [1 : D]$  corresponds to the section in the audio data where bar  $\delta_d$  is played. To calculate such time intervals based on a comparison of the musical content, one needs to provide a local cost measure for comparing short sections of the audio data to short sections of the musical content of the sheet music data. Based on this cost measure, local comparisons can be performed to estimate a start time  $\tau_d$  in the audio recording for each bar  $\delta_d \in \delta$ . We refer to  $\tau := (\tau_1, \dots, \tau_D)$  as the sequence of *audio bar start times*. The time intervals  $I_1, \dots, I_D$  are then identified as

$$I_d = [\tau_d, \tau_{d+1}), \quad d \in [1 : D - 1], \quad (3.1)$$

$$I_D = [\tau_{D-1}, T_{\text{audio}}]. \quad (3.2)$$

For the comparison, it is convenient to have both sheet music and audio data converted to the same domain. Since music is a temporal phenomenon, here, the canonical choice is the time domain. Taking the default bar sequence  $\delta$  into account, and assuming a default tempo, the sheet music data is transformed into a sequence of note events represented by parameters such as onset time and duration in seconds, as well as pitch, see Figure 2.2. Note that to obtain these parameters, first, the onset times and durations are determined in music units such as beats and bars. Then, the tempo parameter assumed for the score is used to convert these musical units to physical units of time such as minutes and seconds. We discuss the peculiarities of the required transformations in Chapter 5. At this point, it is only important to note that we need to keep track of which note events belong to which bars, so later we

can find the correct bar from a given note event. The start and end times of bars, which we are interested in, can directly be derived from the note event onset times by keeping book of which note events belong to which bar. If bar  $d$  starts with a rest such there is no note event whose start time marks the start time of the bar, one can use the offset time of the last note event in the previous bar  $d - 1$  instead. If, additionally, bar  $d - 1$  ends with a rest, the start time of bar  $d$  must be estimated from the surrounding note events and the durations of the rests. Let  $T_{\text{score}}$  be the duration of the note event sequence in seconds. Then we define the *score bar start times*  $\sigma = (\sigma_1, \dots, \sigma_D), \sigma_d \in [0, T_{\text{score}}], \sigma_1 \leq \dots \leq \sigma_D$  such that time  $\sigma_d$  corresponds to the start of bar  $\delta_d$  in the note event sequence.

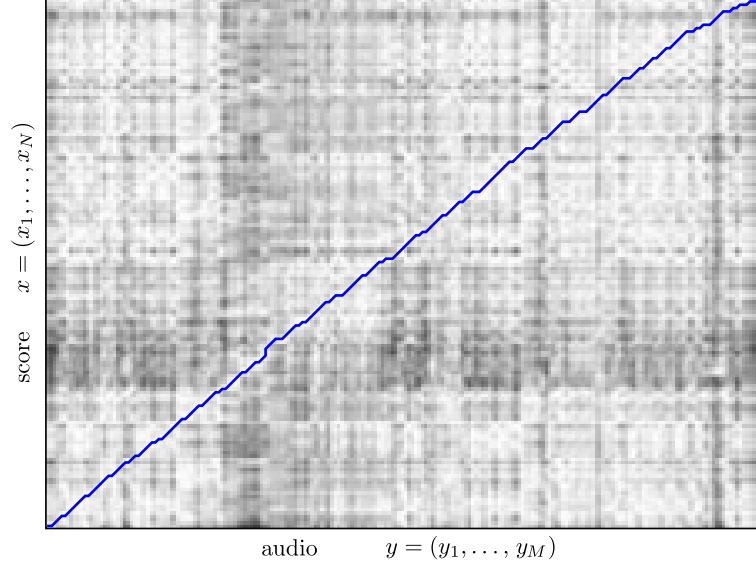
Having obtained a sequence of note events from the sheet music, the remaining task could be called *note event-audio synchronization*, but often it is called *MIDI-audio* or *score-audio synchronization* instead. To allow for a local comparison, both the note events and audio data are divided into temporal *frames* with each frame representing a short temporal section of the data. The frames may or may not be chosen to be evenly spaced, to overlap in time, or to be of the same length. Each frame is represented by a feature vector of some feature space. There are several different choices for feature spaces, which can be either close to the audio domain, e.g., spectral features, close to the note event domain, e.g., the note event domain itself, or somewhere in-between like, for example, chroma features (see Section 3.4). Let us call the feature space chosen to represent the frames of note event data  $\mathcal{X}$  and the feature space chosen to represent the frames of the audio data  $\mathcal{Y}$ . The note event data is converted to a sequence of frames  $x = (x_1, \dots, x_N) \in \mathcal{X}^N$ , and the audio data is converted to a sequence of frames  $y = (y_1, \dots, y_M) \in \mathcal{Y}^M$ . For each frame  $y_m, m \in [1 : M]$  obtained from the audio data, the center time position  $\theta_m \in [0, T_{\text{audio}}]$  of the corresponding temporal section is known and stored. Now, each pair of frames  $(x_n, y_m)$  is compared by means of a *local cost measure*  $c : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}_{\geq 0}$ . This measure is supposed to output low values if the input pair of frames is considered similar to each other and high values if it is considered not similar. The resulting cost values can be written as a so-called *local cost matrix* of dimension  $N \times M$ , which is defined by

$$C(n, m) := c(x_n, y_m) \quad (3.3)$$

for  $(n, m) \in Z$ , where  $Z = [1 : N] \times [1 : M]$  is referred to as the set of *cells*. The basic idea, then, is to find a path through the local cost matrix that connects the two beginnings and ends of the feature sequences that is somehow optimal with respect to the local cost along the path, see Figure 3.3. More formally, we define a  $(N, M)$ -*alignment path* as a sequence  $p = (p_1, \dots, p_L)$  with  $p_\ell = (n_\ell, m_\ell) \in Z$  for  $\ell \in [1 : L]$  that satisfies the following properties:

- (P1) The path starts at  $p_1 = (1, 1)$  and ends at  $p_L = (N, M)$  (boundary condition).
- (P2)  $p_\ell - p_{\ell-1} \in \Sigma$  for  $\ell \in [2 : L]$ , with  $\Sigma = \{(0, 1), (1, 0), (1, 1)\}$  being the set of allowed step vectors (step condition).

The step condition (P2) asserts that with each step, the path must move forward by one index position in either only one of the sequences or in both sequences simultaneously. Following the choice of axes depicted in Figure 3.3, this corresponds to making a step towards the right  $(0, 1)$ , upwards  $(1, 0)$ , or diagonally towards the top right  $(1, 1)$ . The step condition, in



**Figure 3.3.** Visualization of a local cost matrix with an optimum alignment path. The path starts at the lower left corner of the local cost matrix and runs towards the upper right corner.

particular, also prevents the path from moving backwards or skipping forward in any of the two sequences.

When given a cost function that assigns each  $(N, M)$ -alignment path  $p$  a cost value  $c(p) \in \mathbb{R}_{\geq 0}$ , one can define an *optimum alignment path* as an alignment path with minimum cost over all possible  $(N, M)$ -alignment paths. A typical choice for the cost function, which is used in the dynamic time warping algorithm (DTW) discussed in Section 3.2, is

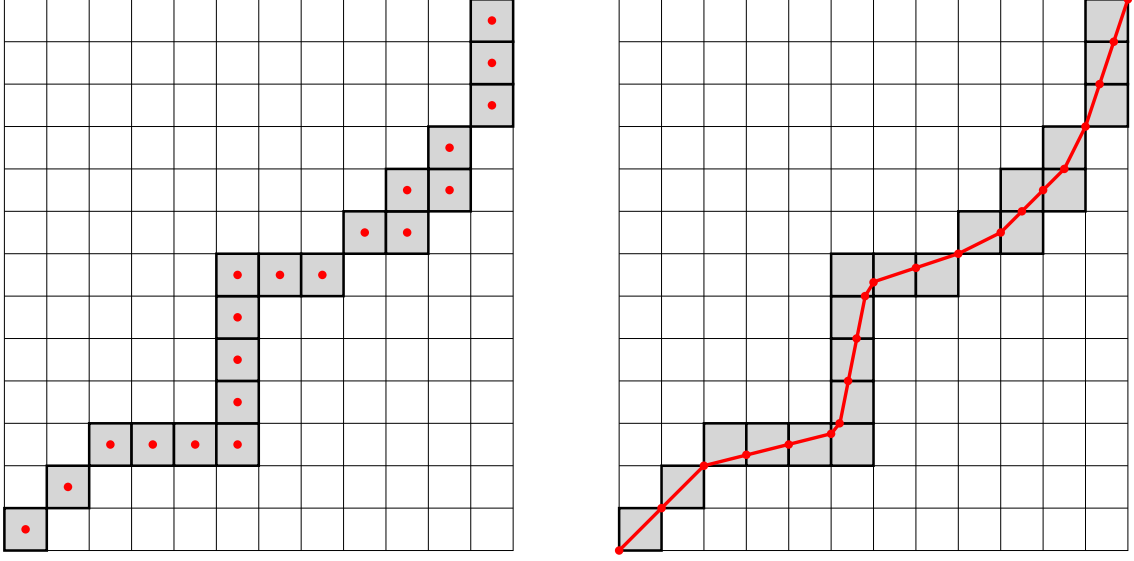
$$c(p) = \sum_{\ell=1}^L C(p_\ell),$$

which is the sum of the local cost covered by the path. However, as we will see in Section 3.3, other choices are possible. Using a suitable cost function, one expects that for all  $(n_\ell, m_\ell)$  of an optimum alignment path, the temporal section of note events represented by frame  $x_{n_\ell}$  in fact corresponds to the temporal section of audio data represented by frame  $y_{m_\ell}$ . An example local cost matrix with an optimum alignment path is depicted in Figure 3.3.

Given an optimum alignment path, it is possible to derive a continuous bijective mapping

$$\omega : [0, T_{score}] \rightarrow [0, T_{audio}] \quad (3.4)$$

using a refinement algorithm and linear interpolation to make the discrete monotonous path continuous and strictly monotonous, see [42] for a detailed description of the algorithm and see Figure 3.4 for an example illustration. Using this mapping, the time  $t$  in the audio recording corresponding to an onset time  $s$  of a note event in the score is simply calculated as  $t = \omega(s)$ . In this way, the sequence of audio bar start times  $\tau := (\tau_1, \dots, \tau_D)$  is calculated as  $\tau = (\omega(\sigma_1), \dots, \omega(\sigma_D))$  and the time intervals  $I_1, \dots, I_D$  can be calculated using Equations 3.1 and 3.2.



**Figure 3.4.** Illustration of the refinement strategy for making a discrete monotonous warping path continuous and strictly monotonous.

There are two commonly used computational approaches to calculating an optimum warping path based on dynamic programming. One approach is called *dynamic time warping*, the other approach uses hidden Markov models (HMMs) and the Viterbi algorithm. Both approaches essentially work the same way, but there are some differences we will point out in the following sections.

## 3.2 Dynamic Time Warping

Dynamic time warping (DTW) is an algorithm that calculates an optimum alignment path based on dynamic programming. It outputs an alignment path  $p$  that minimizes the cost

$$c(p) := \sum_{\ell=1}^L C(p_\ell)$$

over all possible  $(N, M)$ -alignment paths. Given two sequences  $x = (x_1, \dots, x_N) \in \mathcal{X}^N$  and  $y = (y_1, \dots, y_M) \in \mathcal{Y}^M$  and a local cost measure  $c : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}_{\geq 0}$ , the local cost matrix  $C$  is computed as  $C(n, m) = c(x_n, y_m)$  for all  $(n, m) \in Z$ . Then, a so-called *accumulated cost matrix*  $D \in \mathbb{R}_{\geq 0}^{N \times M}$  and a so-called *optimum predecessor position matrix*  $P \in Z^{N \times M}$  are computed as follows. First, we initialize the starting position in the accumulated cost matrix with the local cost at the starting position:

$$D(1, 1) = C(1, 1). \quad (3.5)$$

Then, for all remaining positions  $(n, m) \in Z \setminus \{(1, 1)\}$  we perform the following steps.

- Determine the set  $Z_{n,m} \subset [1 : N] \times [1 : M]$  of matrix positions from which the position  $(n, m)$  can be reached in a single step with any step vector  $\varsigma \in \Sigma$ . We also call this the set of possible predecessor positions:

$$Z_{n,m} = \{(n, m) - \varsigma \mid \varsigma \in \Sigma\} \cap Z \quad (3.6)$$

- Determine the optimum predecessor position at position  $(n, m)$ :

$$P(n, m) = \arg \min_{z \in Z_{n,m}} [D(z)]. \quad (3.7)$$

Note that the accumulated cost  $D(z)$  must have already been calculated for all  $z \in Z_{n,m}$ , which can be achieved by performing the computations in a column-wise or row-wise fashion from the bottom left to the top right.

- Calculate the accumulated cost at position  $(n, m)$ :

$$D(n, m) = C(n, m) + D(P(n, m)). \quad (3.8)$$

At cell  $(n, m)$ , the accumulated cost matrix  $D$  stores the minimum cost of any alignment path starting at position  $(1, 1)$  and ending at position  $(n, m)$  using the allowed step vectors. From the optimum predecessor position matrix  $P$ , the output optimum alignment path is calculated by backtracking from the end position  $(N, M)$  until the start position  $(1, 1)$  is reached:

$$p_L = (n_L, m_L) = (N, M) \quad (3.9)$$

$$p_\ell = P(n_{\ell+1}, m_{\ell+1}), \quad \ell = L - 1, \dots, 1. \quad (3.10)$$

Note that the output optimum alignment path is not necessarily unique, because the  $\arg \min$  in (3.7) might not always be unique. The recursive calculation of  $D$  can be thought of as stepping through each position  $(n, m)$  of the local cost matrix and then checking which one of the three possible predecessor positions in the path, i.e.,  $(n - 1, m)$ ,  $(n, m - 1)$ , and  $(n - 1, m - 1)$ , has the lowest accumulated cost and, thus, leads to the lowest accumulated cost for the current position.

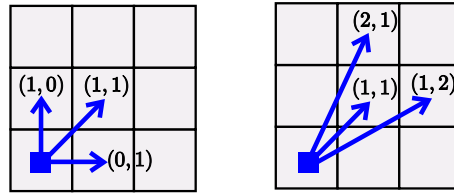
Instead of using a cost measure and minimizing cost, one can use a similarity measure and maximize similarity. Furthermore, the dynamic time warping algorithm can be modified in several ways. A common technique to reward or penalize certain step vectors is to introduce so-called *step weights*. Here, each step vector  $\varsigma \in \Sigma$  is associated a step weight  $w(\varsigma) \in \mathbb{R}_{\geq 0}$ , and Equations (3.7) and (3.8) are modified to

$$P(n, m) = \arg \min_{z \in Z_{n,m}} [w((n, m) - z) \cdot C_{n,m} + D(z)], \quad (3.11)$$

$$D(n, m) = w((n, m) - P(n, m)) \cdot C(n, m) + D(P(n, m)). \quad (3.12)$$

Instead of using the classic set of step vectors  $\Sigma = \{(1, 0), (0, 1), (1, 1)\}$ , a different set can be chosen. For example, the choice  $\Sigma = \{(2, 1), (1, 2), (1, 1)\}$  constrains the path to follow the diagonal direction more closely, see Figure 3.5. Please note that when altering the set





**Figure 3.5.** Two possible choices for the set of step vectors  $\Sigma$  in the DTW algorithm. Left:  $\Sigma = \{(1, 0), (0, 1), (1, 1)\}$ , right:  $\Sigma = \{(2, 1), (1, 2), (1, 1)\}$ .

of step vectors one has to take into consideration that this may cause some of the cells in the accumulated cost matrix to not be reachable by any valid alignment path. This may require some minor adjustments in the described computations we will not further discuss in the following.

In case of using a cost measure and minimizing cost, it is possible to add a positive cost offset to the local cost. This has the effect of making each step more expensive by the same amount, which, in turn, leads to shorter paths, i.e., paths with less steps taken, being preferred over longer paths with less average cost. Alternatively, one can think of this as reducing the contrast between low-cost and high-cost passages, reducing the relative cost for overcoming a high-cost passage compared to taking a longer path around that passage.

The computational complexity and memory complexity of the DTW algorithm are in  $\mathcal{O}(N \cdot M)$ , which, for  $N = M$  makes them quadratic in the input sequence length  $N$ . In classical music, audio recordings can easily exceed a duration of 20 minutes. Assuming a frame rate of 10Hz for the input sequences and a duration of 20 minutes, the amount of matrix cells to be computed would be  $(20 \cdot 60 \cdot 10)^2 = 144,000,000$ . A multi-scale approach can be used to significantly reduce the complexity of the algorithm [100]. Here, the DTW algorithm is first performed on a coarser resolution. Then, the resulting path is used to calculate a constraint region for the matrix of a finer resolution. This process is iterated until the target resolution is reached.

### 3.3 Hidden Markov Models

An optimum alignment path as defined in Section 3.1 can also be computed using the theory of hidden Markov models (HMMs). HMMs are used to model stochastic processes that produce a sequence of observations. The sequence of observations is assumed to be produced by a set of hidden internal states. In this section, we assume that the reader is familiar with the basics of HMMs. For a tutorial on the subject see [118]. Here, we only briefly identify and label the components of an HMM to discuss how it can be used to calculate an optimum alignment path. A HMM is defined by

(H1)  $S = \{S_1, \dots, S_N\}$ , a set of model states,

(H2)  $V$  the set of possible observations,

- (H3)  $A = (a_{i,j}) \in [0, 1]^{N \times N}$ , a row-stochastic matrix, where entry  $(i, j)$  gives the probability of making a transition from state  $S_i$  to  $S_j$  in a single step,
- (H4) for each state  $S_j$ ,  $j \in [1 : N]$  a probability density function  $b_j : V \rightarrow \mathbb{R}_{\geq 0}$  with  $\int_V b_j(v) dv = 1$  that measures how likely it is to observe  $v \in V$  when in state  $S_j$ , and
- (H5)  $\pi = (\pi_i) \in [0, 1]^{1 \times N}$ , a row-stochastic matrix, where entry  $(1, i)$  gives the probability of starting in state  $V_i$ .

Given an observation sequence  $O = (v_1, \dots, v_T) \in V^T$  of length  $T$ , we can use the *Viterbi algorithm* to compute the most likely state sequence  $Q = (S_{q_1}, \dots, S_{q_T})$  that might have led to the given sequence of observations. Similar to the DTW algorithm, here, dynamic programming is used and optimum intermediary results are saved. The subsidiary variables used in the algorithm are

- an accumulated probability matrix  $\delta \in [0, 1]^{N \times T}$  that at position  $(j, t)$  stores the maximum reachable probability of any state sequence  $Q^* = (S_{q_1}, \dots, S_{q_t})$  with  $q_t = j$  leading to the prefix sequence  $O^* = (v_1, \dots, v_t)$  up to index  $t$  of the given observation sequence  $O$ , and
- a matrix  $\psi \in [1 : N]^{N \times T}$  that at position  $(j, t)$  stores the state index  $q_{t-1}$  that maximizes the probability of any state sequence  $Q^* = (S_{q_1}, \dots, S_{q_t})$  with  $q_t = j$  leading to the prefix sequence  $O^* = (v_1, \dots, v_t)$  up to index  $t$  of the given observation sequence  $O$ .

As a first step, the leftmost columns of  $\delta$  and  $\psi$  are initialized. For  $j \in [1 : N]$ :

$$\begin{aligned}\delta_{j,1} &= \pi_j \cdot b_j(v_1), \\ \psi_{j,1} &= 1.\end{aligned}$$

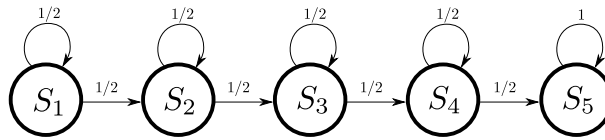
Then, for all subsequent columns, the values of  $\delta$  and  $\psi$  are computed iteratively using the results of the previous column. For  $t \in [2 : T]$ ,  $j \in [1 : N]$ :

$$\begin{aligned}\delta_{j,t} &= \max_{i \in [1:N]} [\delta_{i,t-1} \cdot a_{i,j}] \cdot b_j(v_t), \\ \psi_{j,t} &= \arg \max_{i \in [1:N]} [\delta_{i,t-1} \cdot a_{i,j}].\end{aligned}\tag{3.13}$$

In Equation (3.13), the accumulated probability  $\delta_{j,t}$  for being in state  $S_j$  at observation index  $t$  is computed by first searching for the maximum of the product of accumulated probability at the previous observation index  $\delta_{i,t-1}$  with the transition probability  $a_{i,j}$  of making the step from the previous state  $S_i$  to the current state  $S_j$ , and then multiplying this value by the probability (density)  $b_j(v_t)$  of observing  $v_t$  when in state  $S_j$ .

The optimum state sequence  $Q$  is, then, computed by first searching for the entry in the last column of  $\delta$  with the highest accumulated probability, and then using backtracking until the first column is reached:

$$\begin{aligned}q_T &= \arg \max_{i \in [1:N]} [\delta_{i,T}], \\ q_t &= \psi_{q_{t+1}, t+1}, \quad t = T - 1, \dots, 1.\end{aligned}\tag{3.14}$$



**Figure 3.6.** Illustration of a left-to-right connected HMM (with  $N = 5$  states) for calculating an alignment path.

In the following, we will present a method of calculating an optimum alignment path as defined in Section 3.1 by constructing an HMM and performing the Viterbi algorithm. Note this particular method is chosen only to serve as an example for the concept of calculating an optimum alignment path using HMMs. Different methods might be more useful in practical applications. We start at the point where we are given two sequences  $x = (x_1, \dots, x_N) \in \mathcal{X}^N$  and  $y = (y_1, \dots, y_M) \in \mathcal{Y}^M$  and a similarity measure  $\tilde{c} : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}_{\geq 0}$ . Note that, opposed to our description of the DTW algorithm where we used the symbol  $c$  to denote a local cost measure, we here use the symbol  $\tilde{c}$  to denote a local similarity measure, i.e., high values of  $\tilde{c}$  represent high similarity. As a first step, we construct an HMM from the given data. A set of states  $S$  is created from the sequence  $x$  as follows. For each frame  $x_i$ , we create an associated state  $S_i$  such that we get  $N$  distinct states, even if  $x_i = x_j$  for some  $i, j \in [1 : N]$ . Next, we identify the set of possible observations  $V$  with the feature space  $\mathcal{Y}$  by setting  $V = \mathcal{Y}$ . Furthermore, we consider the sequence  $y$  as our observation, i.e.,  $O = (y_1, \dots, y_M)$ . Note that for this choice, the length of the observation sequence  $T$  is simply the length of the feature sequence  $y$ , i.e.,  $T = M$ . Since property (P2) of the alignment path prohibits backward steps and restricts forward steps to a step size of 1, we choose the state transition matrix  $A$  as follows:

$$a_{i,j} = \begin{cases} 1 & \text{if } j = i = N \\ 1/2 & \text{if } j = i \neq N \\ 1/2 & \text{if } j = i + 1 \\ 0 & \text{otherwise.} \end{cases} \quad (3.15)$$

This leads to a so-called left-to-right connected HMM, see also Figure 3.6. In our case, we only allow transitions from a state  $S_i$  to itself or, if exists, to the subsequent state  $S_{i+1}$ . Since, at this time, we do not want to prefer any of the transitions, we simply choose to evenly distribute the probability over all outgoing transitions for each state.

To create the observation probability density functions  $b_j, j \in [1 : N]$ , we can make use of the given similarity measure  $\tilde{c}$ , but we need to be careful about the requirement that  $\int_V b_j(v) dv = 1$ . This can, for example, be achieved by a normalization

$$b_j(v) = \frac{\tilde{c}(x_j, v)}{\int_{\mathcal{Y}} \tilde{c}(x_j, w) dw} \quad (3.16)$$

assuming that the integral in the denominator exists and is greater than zero. In practice, however, these probability distributions are often obtained through machine learning techniques by means of given training data. The availability of suitable training algorithms is in fact one of the major advantages of the HMM approach over the DTW approach.

Now, we can compute a local similarity matrix

$$\tilde{C}(n, m) = b_n(y_m). \quad (3.17)$$

To make sure that the output alignment path starts at position  $(1, 1)$ , which is required by property (P1), we choose  $\pi$  as

$$\pi_i = \begin{cases} 1 & \text{if } i = 1 \\ 0 & \text{otherwise.} \end{cases} \quad (3.18)$$

Now we can rewrite the Viterbi algorithm for the special case of our HMM created above. We use a notation that is similar to the one we used for the DTW algorithm, so that we can more easily compare the two. In particular, instead of  $\delta$  we use a matrix  $\tilde{D}$  and instead of  $\psi$  we use a matrix  $\tilde{P}$ . The initialization is a bit different from the DTW case. Because we have set the start distribution  $\pi$  to take the value 1 for  $i = 1$  and 0 otherwise, in the first column of  $\tilde{D}$ , all values become 0 except for the position  $(1, 1)$ . The first column of the optimum predecessor position matrix  $\tilde{P}$  is initialized with values  $(1, 1)$ , but these values will not be used in any later step.

$$\begin{aligned} \tilde{D}(1, 1) &= \tilde{C}(1, 1), \\ \tilde{D}(n, 1) &= 0, \quad n \in [2 : N], \\ \tilde{P}(n, 1) &= (1, 1), \quad n \in [1 : N]. \end{aligned}$$

Looking at Equation (3.13) and using our allocation of  $A$  from Equation (3.15), we see that the set of allowed step vectors, i.e., the step vectors with transition probability  $a_{i,j} > 0$ , in the case of our HMM is  $\Sigma = \{(0, 1), (1, 1)\}$  for all column positions  $t \in [1 : M - 1]$ . With each step, we can either stay in the same state or advance to the next state, but the observation index  $t$  always increases by 1. For all remaining columns  $(n, m) \in [1 : N] \times [2 : M]$ , the following steps are performed.

- Determine the set  $Z_{n,m} \subset [1 : N] \times [1 : M]$  of possible predecessor positions. This is the same as in the DTW case in Equation (3.6):

$$Z_{n,m} = \{(n, m) - \varsigma \mid \varsigma \in \Sigma\} \cap Z \quad (3.19)$$

- Determine the step predecessor position  $\tilde{P}(n, m) \in Z_{n,m}$  that leads to the maximum accumulated probability when arriving at position  $(n, m)$ . In comparison with Equation (3.13), we can leave out the transition probabilities  $a_{i,j}$  since they all have the same value, i.e., 0.5, and, therefore, have no effect on finding the maximum.

$$\tilde{P}(n, m) = \arg \max_{z \in Z_{n,m}} [\tilde{D}(z) \cdot 0.5] = \arg \max_{z \in Z_{n,m}} [\tilde{D}(z)]. \quad (3.20)$$

- Calculate the accumulated probability at position  $(n, m)$ . Note that, unlike in Equation (3.8), here, the local similarity is multiplied instead of added to the accumulated cost:

$$\tilde{D}(n, m) = \tilde{C}(n, m) \cdot \tilde{D}(\tilde{P}(n, m)) \cdot 0.5. \quad (3.21)$$

For the backtracking to retrieve the output path  $p$ , unlike in Equation (3.14), we choose the fixed end state  $N$  instead of searching for the state  $i \in [1 : N]$  that leads to the highest total probability. We do this to assert that the output path  $p$  ends at position  $(N, M)$  as is required by property (P1). This makes the backtracking procedure the same as in the DTW case:

$$p_L = (n_L, m_L) = (N, M) \quad (3.22)$$

$$p_\ell = \tilde{P}(n_{\ell+1}, m_{\ell+1}), \quad \ell = L - 1, \dots, 1. \quad (3.23)$$

From the comparison, we see that the algorithm for calculating an optimum alignment path from our HMM and the DTW approach essentially work in the same way. The differences we identified are:

- Comparing the set of allowed step vectors in the HMM approach to the one in the DTW approach, we see that, in the case of the HMM we have constructed, the step vector  $(1, 0)$  is missing. This is due to the fact that in the HMM scenario, by design, the position in the observation sequence must do a single step forward in every iteration. This also leads to  $L = M (= T)$ , i.e., the length of the path is equal to the length of the observation sequence.
- In contrast to the cost measure  $c$  in the DTW algorithm, in the HMM scenario, the similarity measure for the local comparison  $\tilde{c}$  is required to be (or must be altered to become) a probability density function for each  $x_n, n \in [1 : N]$ , i.e.,  $\int_{\mathcal{Y}} \tilde{c}(x_n, v) dv = 1$  for all  $n \in [1 : N]$ .
- Comparing Equation (3.21) to Equation (3.8), we see that instead of adding the local cost  $C(n, m)$  to the accumulated cost  $D(z)$ , the local observation probability  $\tilde{C}(n, m)$  is multiplied to the accumulated probability  $\tilde{D}(z)$ . Based on this, we identify the alignment path cost function that is optimized by the described HMM approach as the product of the local observation probabilities along the path

$$c(p) = \prod_{\ell=1}^L \tilde{c}(x_{n_\ell}, y_{m_\ell}).$$

However, what is often done in practice is not to maximize the probability, but, equivalently, the logarithm of the probability. In that case, the product is replaced by a sum which makes the term that is optimized for look very similar to the DTW case:

$$\log[c(p)] = \log\left[\prod_{\ell=1}^L \tilde{c}(x_{n_\ell}, y_{m_\ell})\right] = \sum_{\ell=1}^L \log[\tilde{c}(x_{n_\ell}, y_{m_\ell})].$$

Unlike in DTW, which is a symmetric approach handling both input sequences the same way, the HMM approach is asymmetric in that the observation sequence is handled differently from the state sequence. We will get back to this asymmetric behavior and its advantages in Chapter 7 when dealing with structural differences between the input sequences.

In the DTW approach, we introduced step weights to reward or penalize certain step vectors. In the HMM approach, we can influence the preference of certain step vectors by making

modifications to the state transition probabilities  $a_{i,j}$ . Then, Equations (3.20) and (3.21) change to

$$\tilde{P}(n, m) = \arg \max_{z \in Z_{n,m}} \left[ \tilde{D}(z) \cdot a_{z_{(1)},n} \right], \quad (3.24)$$

$$\tilde{D}(n, m) = \tilde{C}(n, m) \cdot \tilde{D}(\tilde{P}(n, m)) \cdot a_{\tilde{P}(n, m)_{(1)},n}, \quad (3.25)$$

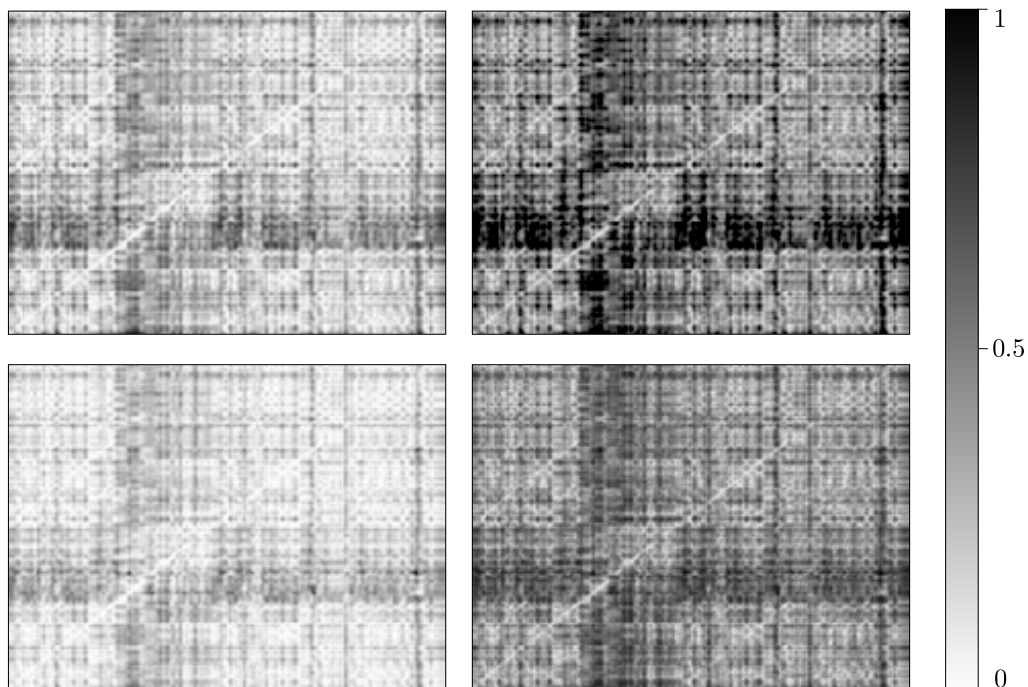
with  $z_{(1)}$  denoting the first component of the vector  $z$  and  $\tilde{P}(n, m)_{(1)}$  denoting the first component of the vector  $\tilde{P}(n, m)$ . In contrast to the DTW approach, the state transition probabilities  $a_{i,j}$  are used as a multiplicative factor affecting the accumulated probability instead of the local cost.

### 3.4 Local Similarity and Mid-Level Representations

The dynamic programming approaches discussed in the previous sections perform a global optimization based on results of local comparisons given as a local cost/similarity matrix. The output of the local comparisons strongly depends on the choice of features and the similarity measure. There are many possible choices for features and similarity measures. The choice of features determines which aspects of the music data are considered in the comparison. The similarity measure determines in what way and to what degree a pair of features is considered similar. The better the chosen features and similarity measure capture the intended notion of similarity, the more reliably the synchronization output of the global optimization will deliver the intended results. Furthermore, the clearer they manage to separate similar from dissimilar, the more robust the results will be.

The differences caused by different choices of features and similarity measures can be discussed by looking at greyscale image visualizations of the cost/similarity matrices they produce, see Figure 3.7. Assuming that the cost/similarity measure is normalized to output values between 0 and 1, here, the cost/similarity value of each matrix cell is visualized as a greyscale value with black representing the value 0 and white representing the value 1. Let us assume that we are given some kind of ground truth similarity measure that reflects an intended notion of similarity by human judgement. This gives a ground truth picture that probably shows some kind of pattern. Each choice of features and similarity measure also produces a picture. If the choice does not capture the right aspects of the data, the pattern will look different from the one in the ground truth picture. Otherwise, the remaining differences will manifest as differences in brightness (a global cost/similarity offset), contrast (shape of the dynamics curve that maps cost/similarity values to greyscale values), sharpness (separation between similar and not similar), and noise. Noise can be caused by noise in the input data itself or by unintended systematic aspects of the data that have not been eliminated in the comparison. Depending on the optimization criterion, i.e., the cost function of the alignment path, the results of the optimization process may be more or less sensitive to such differences.

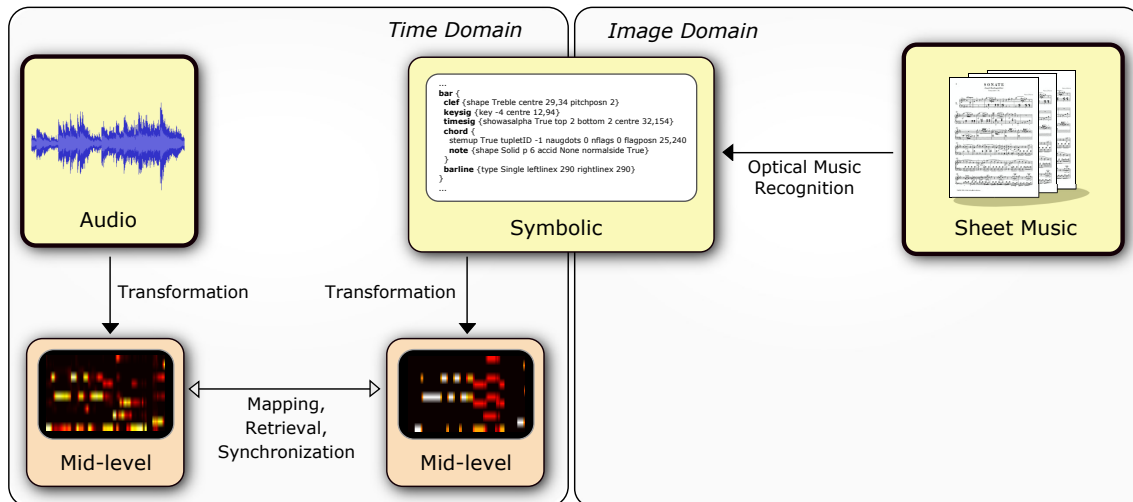
For our scenario of sheet music-audio synchronization, our intended notion of local similarity is to consider a frame derived from audio data similar to a frame derived from sheet music data when the corresponding audio section is a performance of the corresponding sheet music



**Figure 3.7.** Comparison of visualizations of local similarity matrices obtained using different choices for the local similarity measure.

section. Due to possible OMR errors and the many degrees of freedom in the transformations from symbolic score data to audio data (in our case, interpretation, performance, and recording, see Chapter 2), this notion of similarity is inherently somewhat fuzzy. Finding good features and similarity measures is hard because the aspects of the data that are important for the intended similarity consideration may change depending on the content even within a single piece of music. For example, at a section of rapidly varying pitches, a suitable similarity consideration might be based on the aspect of pitch. However, at a section that is governed by rhythmic patterns with constant pitches, a suitable similarity consideration might require to be based on the aspect of rhythm instead. Even though one might simply choose to consider both aspects simultaneously, as done in [43], the more aspects one requires to be similar simultaneously, the less tolerance there can be left for differences that are inherent in the data. Without a mechanism that decides when to consider which aspects, one has to choose a tradeoff between the amount of aspects to consider and the amount of tolerance for inherent differences of the data in the similarity comparison.

It seems clear that audio data and note events are so different in both the information they carry and their level of abstractness, that they cannot be compared directly (see Chapter 2). Therefore, a canonical approach might be to first transform both data types to either audio data or note events using transcription or synthesis, review Figure 2.2, and then perform the local comparison on the single target data type. For both target data types, however, this approach has a disadvantage. Audio transcription of polyphonic music is a very hard task that requires making hard decisions about the existence and the exact parameters of note events. Especially for music involving big ensembles of voices and instruments with soft onsets,



**Figure 3.8.** Illustration of the comparison of sheet music and audio data via mid-level features.

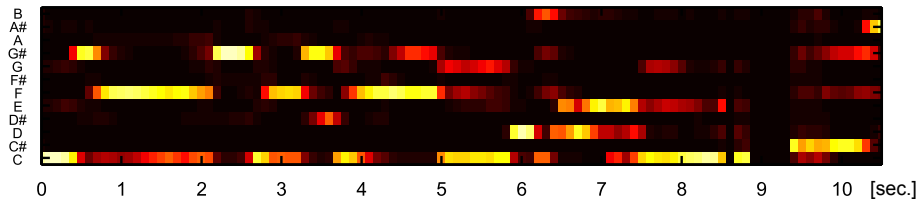
results of automatic transcription are unsatisfactory. To avoid that the local comparison of the transcription results with the note events severely suffers from the transcription errors, one would have to introduce a significant level of fuzziness to the cost measure. This effectively annihilates the advantage of having a simple class of symbolic events to compare.

The disadvantage of using synthesis to transform the note events to audio data is that one has to decide on a single synthesis result even though there are large variations on how the notes of the piece of music might be realized in a concrete performance and recording. If the synthesis does not match the performance and recording well, a local cost measure on the audio data (e.g. as spectral frames) cannot be expected to give very accurate results either. Even though the above approaches may be reasonable choices in some scenarios, the hard decisions that have to be made in either transcription or synthesis seem to be unnecessary for the somewhat easier task of finding the temporal correspondence between two given representations.

A different approach that avoids making such hard decisions is to train a model for how note events might manifest in the audio domain if they would be performed. In contrast to performing synthesis and, hence, deciding on a single realization, here, the model may be fuzzy enough to cover different performances and playing styles. Furthermore, in cases where special constraints on the type of music, scores, and performances are given, these models can as well be optimized for the special situation and, thus, benefit from the additional knowledge. In this approach, the complexity of the task is shifted towards both designing suitable models and training these models. Having a good model that suits the data well, one can obtain very accurate results. However, finding models that work well for a wide range of instrumentations and styles is, again, very hard.

The approach taken in this thesis is to use a so-called *mid-level representation*, i.e., a representation of the music data that is somewhere between audio data and symbolic data, see Figure 3.8. The idea behind mid-level representations is to eliminate certain aspects such as timbre, tuning, or loudness as much as possible, but to avoid making hard decisions on the ex-





**Figure 3.9.** Illustration of chroma features. The diagram shows a visualization of chroma features for approximately 10 seconds of piano music. The horizontal axis represents time in seconds. The vertical axis is divided into 12 chroma classes labeled C, C $\sharp$ , D, . . . , B. Color values represent the intensity of a chroma at a given time with black indicating low intensity, red indicating medium intensity, and yellow/white indicating high intensity.

istence and parameters of symbolic events. After converting two digital music representations to such a mid-level representation, a comparison between the two can be performed that is independent of the eliminated aspects. Since the aspect of timbre is one major source of differences between the note events and the audio data we want to compare, timbre-independent mid-level representations are particularly useful in our scenario. Using a timbre-independent representation for the comparison, one expects to achieve some robustness against differences in instrumentation, interpretation and performance.

In recent years, *chroma features* have become a very popular type of mid-level representation for the comparison of music data [51, 11, 8, 97, 54, 148, 133, 95]. Here, the term *chroma* refers to the 12 pitch classes of the equal-tempered chromatic scale in Western music, commonly labeled C, C $\sharp$ , D, . . . , B. A chroma vector  $f \in \mathbb{R}_{\geq 0}^{12}$  represents the distribution of energy among the chroma classes for a given temporal section of the data, see Figure 3.9. A comparison between two chroma vectors  $f$  and  $g$  can then be done using simple measures such as

$$\tilde{c}_{\text{cosine}}(f, g) = \frac{\langle f, g \rangle}{\|f\| \cdot \|g\|}$$

which is a similarity measure that produces values in  $[0, 1]$  and can be thought of the cosine of the angle of the two vectors in 12-dimensional euclidean space. If the two vectors point roughly in the same direction, i.e., if the relative strength of the 12 pitch classes roughly match, this measure outputs values close to 1, indicating high similarity, and if the vectors are more or less orthogonal, this measure outputs values close to 0, indicating low similarity. From this, a cost measure  $c_{\text{cosine}}$  can easily be constructed by setting

$$c_{\text{cosine}}(f, g) = 1 - \tilde{c}_{\text{cosine}}(f, g) = 1 - \frac{\langle f, g \rangle}{\|f\| \cdot \|g\|}.$$

Obtaining a chroma representation from a sequence of note events is a rather straightforward process. It can be done by simply identifying the pitch class of each note event and distributing energy among the 12 classes according to which note events lie within the given temporal section. More challenging is the extraction of chroma features from the audio data. In fact, in contrast to the approaches for the comparison of note events and audio data described above, here, the complexity of the comparison task is shifted towards the process of the feature extraction from the audio data. Several approaches have been described in the literature

incorporating ideas for eliminating timbre, tuning, loudness, agogics, and transients from note onsets.

In our scenario of comparing note event data with audio data, both data types are converted to chroma features, i.e.,  $\mathcal{X} = \mathcal{Y} = \mathbb{R}^{12}$ . Note that chroma features can also be created for other musical scales, but since in our scenario the dataset consists of classical Western music only, we always assume the equal-tempered scale with 12 chroma. A general limitation of chroma features is that they only work well for pitched music, and they only localize well if the pitch content of the music is not stationary for too long. For percussive music or parts with longer sections of stationary pitch content, a different class of features must be used to complement the characteristics of the chroma features.

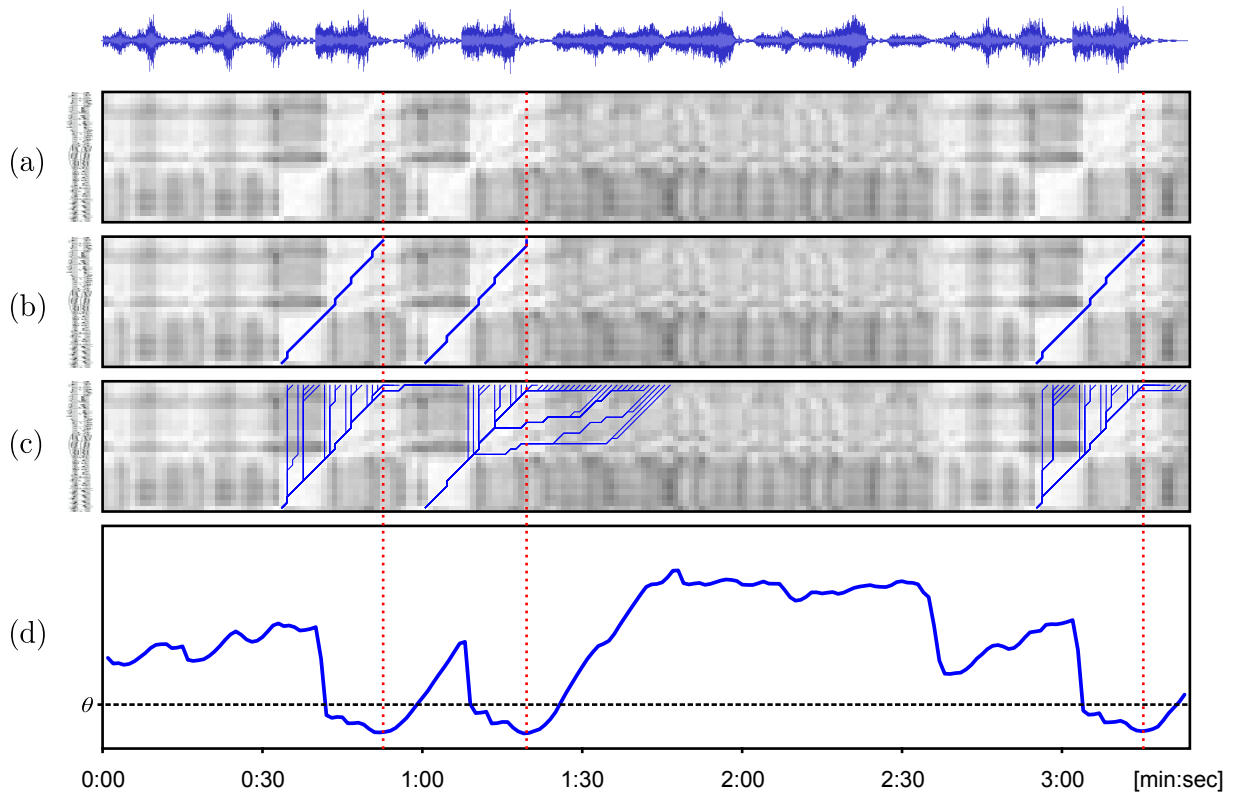
### 3.5 Subsequence Dynamic Time Warping

Subsequence dynamic time warping, in the following abbreviated subsequence DTW or SS-DTW, is a variant of DTW that can be used to find subsequences of a sequence  $y \in \mathcal{Y}^M$  that are similar to a (usually shorter) sequence  $x \in \mathcal{X}^N$  [91]. We also refer to this task as *matching*. In the terminology of matching, we denote the sequence  $x$  as the *query* and the sequence  $y$  as the *database* to search in. An example usage would be to search for a sequence of bars from a score in a database of audio recordings, see Figure 3.10. We will make use of this technique in the later chapters of this thesis.

The basic principle behind subsequence DTW is the same as in regular DTW. The difference is that instead of requiring the algorithm to output a single alignment path  $p = (p_1, \dots, p_L)$  with  $p_\ell = (n_\ell, m_\ell) \in Z = [1 : N] \times [1 : M]$  for  $\ell \in [1 : L]$  that starts at  $p_1 = (n_1, m_1) = (1, 1)$  and ends at  $p_L = (n_L, m_L) = (N, M)$ , we try to find locally optimum paths with  $n_1 = 1$  and  $n_L = N$  but without restricting  $m_1$  and  $m_L$ . In the following, we describe the steps performed in the subsequence DTW algorithm using a similar notation as in the description of the regular DTW algorithm in Section 3.2. First, the cost matrix  $C$  is calculated in the same way as for the regular DTW. An example cost matrix is depicted in Figure 3.10(a). To compute the accumulated cost matrix  $D$ , we initialize the complete first row of the accumulated cost matrix with the values of the local cost matrix at the same positions:

$$D(1, m) = C(1, m), \quad m \in [1 : M]. \quad (3.26)$$

For all remaining positions  $(n, m) \in [2 : N] \times [1 : M]$  the accumulated cost matrix is calculated in the same way as in the regular DTW algorithm, see Equations 3.6 to 3.8. Now, instead of starting the backtracking at position  $(N, M)$  and proceeding until we reach position  $(1, 1)$ , we search the  $N$ -th row (the top row in the illustrations) of the accumulated cost matrix for local minima that lie below a certain cost threshold  $\theta$ , and start a backtracking from each of these minima. For each minimum, we terminate the backtracking as soon as we have reached the 1-st row. This way, we get an alignment path for each such local minimum found in the  $N$ -th row of the accumulated cost matrix, see Figure 3.10(b). The values of the  $N$ -th row of the accumulated cost matrix can be plotted as a function of  $m \in [1 : M]$ , see Figure 3.10(d). This so-called *matching curve* reveals the existence of significant minima to the eye of the human observer.



**Figure 3.10.** Illustration of subsequence dynamic time-warping. An excerpt of the score of Beethoven’s Piano Sonata 1 Op. 2 No. 1 Menuetto is used as a query to find occurrences in an audio recording of the complete piece. Vertical dashed lines indicate the end positions of the three occurrences of the queried section in the audio recording. (a) local cost matrix, (b) local cost matrix with three matches, (c) illustration of groups of paths that start at the same position, (d) matching curve.

Let us assume we get a total of  $R$  alignment paths, which we write as a set  $H(x, y) = \{p^1, \dots, p^R\}$ . We also refer to these paths as *matches* and to  $c(p) = D(n_L, m_L) = D(N, m_L) = \sum_{\ell=1}^L C(n_\ell, m_\ell)$  as the cost of the match  $p$ . The amount of matches obtained can be controlled by the choice of the cost threshold  $\theta$ , which can be thought of as an upper bound for the cost of a path  $p$  to be counted as a match. To avoid getting many possibly overlapping matches inside regions of low cost near the actual occurrences of the query in the database, different strategies can be pursued. A common strategy is to ignore all local minima that lie inside a certain neighborhood of another local minimum with less cost. This can be achieved by iteratively searching for the global minimum and then removing the minimum and its neighborhood from the search space until no more minimum below the threshold  $\theta$  is found. Here, it seems to be reasonable to choose the size of the neighborhood depending on the length of the query. An alternative approach is to determine the neighborhood to remove around a local minimum as the set of positions  $m$  in the top row of the accumulated cost matrix that lead to the same start position  $p_1$  when backtracking is started at position  $(N, m)$ . This situation is depicted in Figure 3.10(c). The illustration shows that the paths that lead to the same start position branch outwards from the path of minimum cost in a tree-like structure.

From each match  $p \in H(x, y)$  one can compute a corresponding temporal region  $T(p) = T = [t_1, t_2]$  in seconds in the audio data. To this end, we first compute the interval  $F$  of index positions in the audio feature sequence that is covered by the path  $p$ . Recalling that  $p = (p_1, \dots, p_L)$  with  $p_\ell = (n_\ell, m_\ell)$ , the interval  $F$  is simply determined as

$$F = [m_1, m_L]. \quad (3.27)$$

With  $r$  denoting the temporal resolution of the feature sequence in Hertz, the time interval  $T$  in seconds is computed as

$$T = [t_1, t_2] = \left[ \frac{m_1}{r}, \frac{m_L}{r} \right]. \quad (3.28)$$

The match  $p$  can, then, be interpreted as saying that the region  $T$  in the audio data is similar to the sequence of bars that were used as the query in terms of musical content. One might, furthermore, use this similarity in content as an indicator for concluding that the temporal region  $T$  is an acoustic realization of the queried sequence of bars. We will make use of this technique in Chapter 6 to segment and identify the contents of sheet music books based on a content-based comparison to audio recordings. However, since pieces of music may contain bar sequences with similar content, a content-based similarity does not guarantee that a match always indicates an occurrence of the queried bar sequence in the audio data. More on such possible confusion can be found in [45].

Since subsequence DTW uses the same basic principles as regular DTW, but with less constraints, its results are more sensitive to factors such as noise, e.g., OMR errors, or the effect of different settings for the choice of features, cost measures, step vectors or step weights. This sensitivity makes subsequence DTW a suitable instrument for measuring such effects in experiments, as has been done in [48, 45].

## Chapter 4

# PROBADO Music Document Organization System

In the previous chapter, we have discussed how a basic synchronization of sheet music and audio data can be computed and how sections of sheet music can be matched against a database of audio recordings. But, so far, we always have assumed “friendly” conditions as a basis for being able to perform our algorithms. In this chapter we will make the transition to the real-world scenario of a digital music library project, where, as we will see, the conditions are not so friendly at all. We identify the additional challenges that need to be solved and propose an overall framework and workflow for solving these challenges in the context of a document organization system for a digital music library. In Section 4.1 we give a brief introduction to the PROBADO Music Project including some relevant details on the database scheme to store metadata of music documents. In Section 4.2, we identify and review the real-world challenges that distinguish the situation in a real-world digital music library from the “friendly” conditions assumed in the previous chapter. We present our document organization framework for building up and maintaining a digital music library in Section 4.3. Finally, a method for handling special cases that make a fully automatic organization of documents very difficult is discussed in Section 4.4.

### 4.1 PROBADO Music Project

The PROBADO<sup>1</sup> project is an initiative to develop a prototypical service and framework for the integration of non-textual digital documents into library workflows and applications [136, 72, 34, 13, 73]. A major goal of the project is to offer content-based retrieval and similarity-based access to non-textual documents similar to what can be offered for text documents already. Besides the development of the general framework and conventions that are designed to work across document types, currently, there are two sub-projects working on the integration of different types of digital documents. One of the sub-projects is the

---

<sup>1</sup><http://www.probado.de/en/home.do.htm>

PROBADO Music project, which is conducted by two project partners. The Bavarian State Library<sup>2</sup> (BSB) located at Munich, Germany, creates, collects, and manages digitizations of music documents and corresponding metadata. The Multimedia Signal Processing Group<sup>3</sup> of the Department of Computer Science III at the University of Bonn, Germany, develops algorithms and tools for the automated organization and semantic linking of the data, for content-based retrieval, as well as for the presentation of and interaction with the content.

The primary document types currently handled in the PROBADO Music project are scanned sheet music books and digitized audio recordings of Western classical music. As of October 2010, the dataset created by the BSB comprises about 50,000 pages of sheet music in 292 books as well as 6485 audio CD tracks with a total duration of about 473 hours. After focussing on classical-romantic piano music in the first phase of the project, now, the repository contains works across all kinds of instrumentations including operas and pieces for symphonic orchestra written by composers such as Mozart, Beethoven, Liszt, Mendelssohn, Schubert, Schumann, and Händel. The digitized sheet music books are available as one scan per page with a resolution of 600 dpi (dots per inch) and a color depth of 1 bit, i.e., black and white. The audio recordings are available as one file per CD track with a quality of 44.1 kHz, 16 bit, stereo. An objective that is followed in building up the collection is to include both sheet music and at least one audio recording for each work that is added.

In addition to the digital music documents, the repository includes a rich set of metadata that is stored in a database. The database scheme used for the metadata is based on the scheme proposed in the Functional Requirements for Bibliographic Records (FRBR) report from the International Federation of Library Associations and Institutions [68]. In this scheme, the data is organized in four main entities called *work*, *expression*, *manifestation*, and *item*. Additionally, information about contributors such as composers, performers, conductors, and publishers is contained. For details regarding the metadata scheme see [34]. In the following, we will only give a brief introduction to the four main entities as far as it is required for the understanding of the remainder of this thesis:

- **Work:** A work entry represents a piece of music in its most abstract form. It represents the intellectual work behind the composition of the piece. Work entries can form hierarchical relationships. For example, there may be work entries for a complete sonata, but there can also be entries for each single movement as well. We refer to work entries that correspond to higher-level works such as concerts, sonatas, or song cycles as *parent-level works*, and to work entries that correspond to individual movements or songs as *leaf-level works*. Examples for leaf-level works entries are *Beethoven, Piano Sonata, Opus 2 No. 1, Allegro* and *Liszt, A Symphony To Dante's "Divina Commedia", I. Inferno*.
- **Expression:** An expression always refers to a particular work and represents either a performance or a transcription of that work. Examples would be a performance of *Piano Sonata Opus 2 No. 1, Allegro, by Ludwig van Beethoven* played by Alfred Brendel or a sheet music transcription of *Liszt, A Symphony To Dante's "Divina Commedia", I. Inferno* created by Breitkopf & Härtel.

---

<sup>2</sup><http://www.bsb-muenchen.de>

<sup>3</sup><http://www-mmdb.iai.uni-bonn.de>

- **Manifestation:** Performances can be captured as audio recordings and then manifest as tracks on an audio CD or CD collection. Transcriptions can manifest as sheet music in a sheet music book. In our case, a manifestation entry represents either a complete audio CD collection or a complete sheet music book. Note that a single sheet music book or audio CD collection can comprise several transcriptions or performances, i.e., expressions of several different works.
- **Item:** An item represents a physical copy of a particular manifestation. In our case, this can be either an audio CD collection or a sheet music book. However, as long as we consider digital music documents only, all copies of a manifestation can be assumed to be identical. Therefore, in this thesis, we omit the item entity and work with manifestations directly.

Given the digital music documents and the corresponding metadata, the PROBADO Music project seeks several goals:

- (G1) Automate the process of adding new content, i.e., digital music documents and metadata, to the repository as much as possible. Keep the required manual effort as low as possible.
- (G2) Perform synchronization (temporal alignment) for all documents that represent the same piece of music.
- (G3) Analyze and index the data to allow efficient content-based retrieval.

The main goal of this thesis, is to achieve goal (G2) for the case of sheet music-audio synchronization. But, as we will see, all three of these goals are strongly interconnected. We will deal with both (G1) and (G2), since (G2) cannot be solved without knowing which documents belong to the same work, which we understand as being part of (G1). Content-based retrieval as demanded in (G3) also plays a role in this thesis, because it can be of great use in the process of solving (G1). Solving both (G1) and (G2) will, for the first time, enable the large-scale synchronization of digital music documents in a digital music library. This allows for rich and novel applications, some of which will be described in Chapter 8. Here, our focus lies on the multimodal presentation of, and navigation in temporally linked sheet music books and audio CD collections enabled by sheet music-audio synchronization. In the application, the users are presented the original sheet music scans and audio recordings. Even though symbolic score data created through OMR is used in the process of organizing the content and calculating the synchronizations, it is never directly presented to the user.

## 4.2 Real-World Challenges

In this section, we want to identify and discuss the challenges that have to be overcome to achieve goal (G2) in the real-world scenario of the PROBADO Music project. First we specify what is given as the starting point in this scenario in more detail.

- **Scanned sheet music books:** A scanned sheet music book is given as a collection of image files with each file representing a single page. All pages of the book including cover and backside are included and numbered in ascending order. The numbering starts with 1, i.e., the cover gets the page number 1. For the range of pages that contain actual score content, which usually excludes some pages at the beginning and end of the book containing publishing information, foreword, and table of contents, the results of optical music recognition performed by SharpEye 2.68 are given as one MRO file per page<sup>4</sup>.
- **Digitized audio CD collections:** A digitized audio CD collection is a set of audio files with each file representing a single track on a CD. All the tracks of all the CDs in the collection are included and assigned two numbers, the disk number in the collection, and the track number on the respective disk. In addition to the audio files, a textfile including information on the track titles is given for each disk. This information is gathered from the Gracenote<sup>5</sup> service. Here, each track title is represented by a single character string with varying format and naming convention.
- **Metadata:** A work entry is given for each song or movement contained in the given sheet music books and audio CD collections. Each work entry includes information such as the name of the composer, a title, and a catalog name and number, e.g. “opus 3 no.2”.

In Chapter 3 we have described how a bar-wise synchronization for a pair of mid-level feature sequences created from sheet music and audio documents can be computed. There, we have made several assumptions asserting somewhat “friendly” conditions. In our real-world scenario, however, many of these assumptions do not hold. We will now give a list of the assumptions made, and discuss in which cases the assumptions holds, in which cases they do not hold, and what are the consequences for reaching our goal (G2).

(A1) *The type of music notation considered is common Western music notation that is printed or electronically typeset (no handwritings).*

This assumption also holds in our real-world scenario, since the data in our repository is only of this type.

(A2) *A pair of sheet music and audio representation for the same single piece of music is given.*

In our scenario, what we are given is scanned sheet music books and audio CD collections. In the terms of the database model, we also refer to these books and collections as manifestations. Each manifestation can possibly comprise many pieces of music. We need to find out, which pieces of music are contained in each given manifestation. Furthermore, we need to determine the start and end location of each piece in the manifestation. In other words, the manifestations have to be segmented into individual pieces of music and each piece of music has to be identified as corresponding to a particular leaf-level work given in the metadata. It turns out that this task is not as simple as it might seem at the first glance. We will discuss this issue in more detail in Section 4.2.1.

---

<sup>4</sup>The SharpEye 2.68 Music Reader software and the MRO files have been introduced in Chapter 2.3

<sup>5</sup><http://www.gracenote.com>





**Figure 4.1.** Example for interrupted bar lines found in the G. Henle edition of Beethoven’s Piano Sonatas Volume 2, Sonata No. 21 in C major, Opus 35, Introduzione (Adagio molto). The SharpEye OMR software recognizes the two bars on the right as a single larger bar, because the bar line is interrupted by the “decresc.” directive. The left bar line is recognized correctly, even though it is also interrupted, but at a different position.



**Figure 4.2.** Example for a complex textual jump directive found in the G. Henle edition of Brahms Scherzo in E $\flat$  minor, Opus 4.

(A3) *The correct set of bar labels  $\mathcal{B}$  of the sheet music is known.*

The set of bar labels is derived from the OMR results and, therefore, directly depends on the quality of the OMR results. Most of the time, bar boundaries are recognized correctly. Typical errors that can happen in the recognition of bar boundaries are bar lines that are missed because they are interrupted by overlapping text or symbols (see Figure 4.1) and note stems that are mistakenly recognized as bar lines. Even though such errors possibly affect many of the bar labels on a single page, they do not pose any significant problem to the sheet music-audio synchronization. For the synchronization, it makes no big difference how exactly the score is divided into bars. A missed bar line, for example, will simply result in a bigger bar whose content will be aligned to the same section of audio data as would be the two original bars.

(A4) *The default bar sequence  $\delta$  as written in the score is known.*

To extract the default bar sequence  $\delta$  from the OMR results, all repeats and jumps in the score have to be correctly recognized and interpreted. Regular repeat signs are correctly recognized at a very high rate. More problematic are repeats with alternative endings, because the brackets for the alternative endings are not recognized by the SharpEye OMR software used in this project. Even more problematic are jump directives that are given as textual annotations in the score. In simple cases, this requires the recognition

Symbol	Elements	Name
$\nu$	$(\nu_1, \dots, \nu_B), \nu_b \in \mathcal{B}$	notation bar sequence
$\delta$	$(\delta_1, \dots, \delta_D), \delta_d \in \mathcal{B}$	default bar sequence
$\delta^*$	$(\delta_1^*, \dots, \delta_{D^*}^*), \delta_d^* \in \mathcal{B}$	estimated default bar sequence
$\pi$	$(\pi_1, \dots, \pi_K), \pi_k \in \mathcal{B} \cup \{\uparrow\}$	performance bar sequence

**Table 4.1.** List of the different bar sequences and their corresponding symbols in this thesis.

and detection of key words such as “da capo”, “d.c.”, “fine” and “coda”. However, textually given jump directive can be much more complex. A good example for this is found in the G. Henle edition of Brahms Scherzo in E $\flat$  minor, Opus 4. Here, the textually given jump directive reads “Da capo lo Scherzo senza ripetizione sin'al segno  $\Phi$  et poi: Trio II”, see Figure 4.2. A correct handling of such a directive, first, requires the correct recognition of all relevant texts and jump markers, i.e., the directive itself, section headings such as “Scherzo” or “Trio II” that might be used as jump targets, and jump marker symbols such as  $\Phi$ . Then, these texts and markers have to be correctly interpreted. The automatic interpretation can be a very hard task, since the text can be in one of many languages such as Italian, German or English. In the SharpEye OMR software, the recognition of textual directives is quite unreliable. In many cases the text is simply ignored, or obstructed from recognition errors. A more detailed study of these issues can be found in [53]. In the scenario of a fully automated processing of the score data, we have to expect errors in the recognized jumps and repeats. This leads to a bar sequence  $\delta^*$  that is possibly different from the correct default bar sequence  $\delta$ . We refer to the sequence  $\delta^*$  as the *estimated default bar sequence*, see also Table 4.1.

(A5) *The audio representation follows the default bar sequence  $\delta$  as written in the score.*

In practice, the given audio recording does not always follow the default bar sequence  $\delta$  as suggested by the given score. Besides the fact that performances may contain sections that are not written in the score, e.g., cadenzas or improvisations, performers might choose to ignore or add repeats, or even introduce shortcuts. Furthermore, the performance might be based on a different edition of the score which uses a different default bar sequence. We assign each audio recording a so-called *performance bar sequence*  $\pi = (\pi_1, \dots, \pi_K), \pi_k \in \mathcal{B} \cup \{\uparrow\}$  that is possibly different from the default bar sequence  $\delta$ . Here, we use the label  $\uparrow$  to mark sections that are not written in the sheet music, e.g., cadenzas. See Table 4.1 for a list of the different bar sequences introduced so far. In our scenario, the performance bar sequence  $\pi$  is unknown. Available, in our scenario, only are the audio recording, the OMR results, and the estimated default bar sequence  $\delta^*$  obtained from the OMR results. Instead of just finding the time intervals  $I_1, \dots, I_D$  according to  $\delta$  as described in Section 3.1, the synchronization task, now, requires us to both determine the performance sequence  $\pi$  and to compute time intervals  $I_1, \dots, I_K$  according to  $\pi$ . This makes the task a lot more complex.

(A6) *The audio representation is played in the same key as the sheet music is written in.*

In practice, in a performance, the key of pieces of music can be transposed to better suit the pitch range of singers or instruments. In such a case, the local comparison using chroma features as described in Section 3.4 will fail to deliver the desired results.

Error	Impact
missing note	only a single note (usually less impact than having a wrong note instead)
missing beam	durations of all notes in the beamed group
wrong/missing accidental	some pitches in the bar
wrong/missing key signature	some pitches on the staff line
wrong/missing clef	all pitches on the staff line
split grand staff	grand staff is mistakenly repeated but with only subsets of the voices
transposing instrument	all pitches of that instrument throughout the whole track

**Table 4.2.** List of common problems in the OMR results together with their projected area of impact.

However, this issue can be solved quite easily. A transposition of the musical key affects timbre-independent chroma features in the same way as a cyclic shift applied on each of the feature vectors. If, for example, a chroma vector  $f = (f_0, f_1, \dots, f_{11})$  is given, applying a cyclic shift of 3 semitones would yield to  $\sigma_3(f) = (f_9, f_{10}, f_{11}, f_0, f_1, \dots, f_8)$ . The correct transposition can be found by trying all 12 possible cyclic shifts and selecting the one that delivers the lowest total cost when performing the DTW algorithm. A drawback of this approach is that the runtime is multiplied by the factor of 12.

(A7) *The note events created from the sheet music are correct regarding pitch, onset time and duration.*

The note events are derived from the OMR results and, therefore, their accuracy strongly depends on the quality of the OMR results. For the type of sheet music considered in this thesis, the OMR results have errors with different levels of impact on the process of deriving note events. Errors with only a local impact, e.g., missing individual accidentals, stems, or notes, can usually be absorbed by the robustness of the global synchronization process. That is because even if there are many local mistakes and inaccuracies, the surrounding correct sections constrain the alignment path to still follow the globally correct direction that connects all these correct sections.

However, other types of OMR errors or missing information can have larger-scale effects on the accuracy of note events. Table 4.2 shows a list of common problems together with their projected area of impact. The problems are ordered by their level of impact with the heavier ones listed towards the bottom. High-impact errors can significantly degrade the quality of the synchronization or even render the results meaningless. An illustration of some typical OMR errors is given in Figure 4.3.

To achieve our goal (G2), the consequences from our discussion of assumptions (A1),(A3) and (A6) do not require further attention at this point. The other issues, however, are more critical and have to be dealt with. All of these critical issues can be grouped into three main challenges that need to be solved to enable the successful sheet music-audio synchronization in our real-world scenario. These will be discussed in the following subsections.

**Figure 4.3.** Comparison of a scanned score and the corresponding results acquired by OMR (excerpt of Beethoven’s Piano Sonata No. 30, Op. 109). Errors in the OMR result are highlighted by shaded boxes. Triangles at the end of a bar indicate that the sum of the durations of the notes does not match the expected duration of the measure. To correct this, SharpEye has “deactivated” several notes which is indicated by greyed-out note heads. Although in this example, the OMR result shows several recognition errors, there is still enough correspondence to allow for a bar-wise synchronization with a corresponding audio recording.

#### 4.2.1 Track Segmentation and Identification

Before we can synchronize sheet music books and audio CD collections, we first need to determine sections therein that correspond to the same leaf-level works. In the following, we will refer to a section in a sheet music book that corresponds to a leaf-level work as a *score track* and to a section in an audio CD collection that corresponds to a leaf-level work as an *audio track*. If not specified whether a track is a score track or an audio track, we use the term *leaf-level track*. We distinguish two subtasks that need to be solved. Firstly, the manifestations have to be segmented into leaf-level tracks. We refer to this task as *track segmentation*. Secondly, for each leaf-level track, we have to determine the corresponding leaf-level work. We refer to this task as *track identification*.

Leaf-level tracks are determined by a start location and end location in the respective manifestation. To further discuss the task of segmenting manifestations into leaf-level tracks, we, therefore, need to define how we want to specify locations in manifestations. In the case of sheet music books, we simply use the bar labels  $\mathcal{B}$  as introduced in Section 3.1 and rewrite them as a string of the form “page\_line\_bar” with *page* being the page number, *line* being the number of the grand staff on the page, and *bar* being the bar number in the grand staff, see Figure 3.2. The tuple 8\_2\_5, for example, denotes the 5-th bar in the 2-nd grand staff on the 8-th page. Since our goal is a bar-wise synchronization, we don’t need to be able to specify locations in sheet music books more accurately than on the level of individual bars.

The image displays two systems of musical notation for Beethoven's Piano Sonata No. 29, Opus 106. The first system, titled "Tempo I", shows a grand staff with piano (p) and pianissimo (pp) dynamics. The second system, titled "Adagio sostenuto (♩ = 92) Appassionato e con molto sentimento" and "Una corda mezza voce", shows a grand staff with a thin-thick bar line at the end of the first section and a special bar line at the end of the second section.

**Figure 4.4.** Example for a regular track segmentation with an indented grand staff from Beethoven’s Piano Sonata No. 29, Opus 106. The ending section ends with a special bar line of type *thin-thick*. The first grand staff of the new section is indented.

In audio CD collections, the disks are usually given in a particular order (otherwise, some arbitrary order is chosen and fixed). Here, the canonical units of segmentation are the disk number and the CD track number on the disk. Using these units, we can specify locations in audio CD collections down to the level of individual CD tracks by writing a location string of the form “`disk_track`”. To be able to specify locations that lie within tracks, we add an identifier of the form “`minutes:seconds.fractions`” for a time position within the track. Thus, the string `3_2_0:47.98` denotes a position at 47.98 seconds in the 2-nd track on the 3-rd disk.

Besides locations, we also want to be able to specify regions in manifestations. A region is specified by a pair of locations, one for the start point and one for the end point. In this context, we use some conventions that make handling regions more convenient. When specifying a region in a score book by two location strings `pageStart_lineStart_barStart` and `pageEnd_lineEnd_barEnd`, by convention, the start bar, the end bar, and all bars the lie in-between belong to the region. Generalizing this concept, we allow location strings that omit some of their components from the end. If such a location with an omitted component acts as the start of a region, our convention is that the omitted component is to be replaced by the smallest valid value. For a location that acts as the end of a region, the convention is that the omitted component is to be replaced by the largest valid value. For example, a region in an audio CD collection specified as `3_2` to `3_2` conveniently describes a region for the complete track 2 on disk 3 without having to know its duration. A region in a sheet music book specified as `8` to `12` describes a region from the first bar in the first grand staff of page 8 to the last bar in the last grandstaff on page 12.

Now, let us get back to the tasks of track segmentation and identification. For the case of audio CDs, a straightforward approach would be to simply use the segmentation that is

The figure displays two systems of musical notation. The first system, starting at measure 23, shows a piano part with intricate rhythmic figures, including triplets and sixteenth-note runs. It concludes with a *dim.* (diminuendo) and *pp* (pianissimo) marking. The second system, starting at measure 27, is titled "Fuge Allegro ma non troppo" and begins with a piano (*p*) dynamic, followed by *sempre p* (piano throughout). The notation includes various fingerings and articulations across both systems.

**Figure 4.5.** Example for an ambiguous track segmentation. The figure shows an excerpt of Beethoven’s Piano Sonata No. 31, Opus 110, at the transition from a section titled “Adagio ma non troppo” to a section titled “Fuge, Allegro ma non troppo”. Even though in the score the former section does not end with a thin-thick barline and the latter section does not start with an indented grand staff, these two sections form two individual CD tracks on a CD collection of Beethoven’s Piano Sonatas performed by Ashkenazy. On the contrary, both sections manifest in just a single CD track on a CD collection by Brendel.

suggested by the CD tracks resulting in regions 1\_1 to 1\_1, 1\_2 to 1\_2, and so on. In the sheet music books, individual songs or movements are usually indicated by indented grand staves and title headings, see Figure 4.4. Let us assume, for a moment, that the locations of indentations and headings are correctly recognized from the sheet music scans, so that we can use them as segment boundaries to specify the regions of score tracks. Now, it may seem that we have already succeeded in segmenting our sheet music books and audio CD collections. However, it turns out that, in practice, different manifestations do not always agree in how to divide pieces of music into individual movements. For example, in one manifestation, a sonata may be divided into three movements, but in another manifestation the same sonata is divided into four movements. A reason for this is that, in many cases, composers did not specify a unique division themselves. Then, the division is subject to interpretation by the editor of the manifestation, and, therefore, it can vary from case to case. See Figure 4.5 for an example. Unfortunately, this happens too often to be ignored in our real-world scenario.

To solve this ambiguity in the division of parent-level works into leaf-level works as suggested by the CD collections and score books, a single unique division needs to be declared as a reference. To this end, we use the set of leaf-level work entries that is given in the metadata and require it to be unique and non-overlapping. Then, the given manifestations have to be segmented according to this set of leaf-level works, and each segment has to be mapped to the corresponding leaf-level work entry in the track identification process. This way, for each leaf-level work entry, we obtain a set of segments of manifestations that are identified as representing the same leaf-level work and, thus, can be mutually synchronized. The framework we propose in Section 4.3 allows to conveniently set, check, and edit the segmentation

of manifestations manually. Approaches to automatically determine the segmentation are discussed in Chapter 6.

### 4.2.2 Structural Differences

Having determined a unique set of leaf-level tracks that can be mutually synchronized, the next practical issue we have to face is structural differences *within* these tracks. As stated in (A5), such differences can originate from differences between the default bar sequence  $\delta$  and the performance sequence  $\pi$ . However, differences also occur if our estimate  $\delta^*$  of  $\delta$  according to (A4) is inaccurate.

The only way to determine the performance sequence  $\pi$  from a given score track and audio track is by comparing their musical content. The same holds for the case of synchronizing two audio recordings with structural differences. Fortunately, sheet music data offers direct access to structural information that is not comprised in audio recordings, e.g., bars, sections delimited by special bar lines, and repeat signs. This additional structural information makes handling structural differences much easier for the case of sheet music-audio than for the case of audio-audio. Approaches to determining the performance bar sequence  $\pi$  based on the comparison of content of a given score track and audio track are proposed in Chapter 7.

### 4.2.3 OMR Quality and Note Events

Since all information about the sheet music documents that can be further processed is parsed from the OMR output, the quality of the OMR results is a crucial factor in the success of our goals. From the OMR results of a complete sheet music book, we need to extract information used for the track segmentation and identification (A2), the detection of repeats and jumps (A4), as well as the deduction of onset times, pitches and durations of note events (A7).

Unfortunately, the OMR software products available at this time do not offer a solution for all of these tasks. In fact, the use case of most of the commercially distributed products seems to focus on people who want to recognize only a few works individually, who do not need the software to understand or handle complex jumps, and who are willing to put effort into manually correcting the OMR errors using a graphical user interface. The recognition and interpretation of higher-level semantics such as the segmentation of the material into individual works with associated titles, complex repeats and jumps, textually specified tempo directives, and transposing instruments is not provided by these products.

The OMR software leaves us with two types of shortcomings. The first type is information that is inaccurate, e.g., wrong accidentals, defective text, or erroneously split grand staves. The second type is information that is missing, which, in our scenario accounts for the majority of cases. Examples for information that is important in our scenario, but that is not output by the OMR are missed out symbols such as accidentals, clefs, beams, and notes, missed out text elements, as well as higher-level information that is completely missing such as tempo, complex repeats and jumps, and transposing instruments (review Section 2.3). These shortcomings

L. van Beethoven, Op. 68.  
Bearbeitung von Franz Liszt.

**Allegro ma non troppo.**  $\text{♩} = 68.$

Viol.  
*p*  
Bratschen u. Vcl.  
Ped. \*

**Figure 4.6.** Example for a tempo directive that is given both as text element as well as symbolic metronome scale. The example is found at the beginning of the piano transcription by Franz Liszt of Beethoven’s Symphony No. 6, Opus 68, “Pastorale”, Breitkopf & Härtel.

affect all three tasks stated in (A2), (A4), and (A7). However, while (A2) and (A4) concern the structure of the musical content represented by note events, (A7) concerns the content itself. Since solutions to questions concerning structure can be computed or supported by the comparison of the musical content, it is particularly important to extract the content as accurately as possible. In our scenario, content-based comparison is performed on the level of chroma-based features (review Section 3.4). Therefore, in this scenario, the most critical aspects of the content are the pitches and relative onset times of note events.

As listed in Table 4.2, the problems with the highest impact on the accuracy of the content are transposing instruments, erroneously split grand staves, missing clefs, and wrong key signatures. It turns out that the amount of problems of these types in the OMR results strongly depend on the complexity of the underlying scores. While for piano music and smaller ensembles, these problems are little enough to still allow for a reasonable synchronization, for orchestral scores, the distortion of the content is often large enough to render the results of the synchronization approach described in Chapter 3 useless for our application. In addition to the problems listed in Table 4.2, the missing information about the tempo can also have a negative effect on our content-based comparison. Similar to the textual jump directives discussed in (A4), tempo directives in the sheet music are often written as text element in Italian, German or a different language. In more modern score editions, tempo directives are also often given using a symbolic metronome scale as depicted in Figure 4.6. Unfortunately, in our case, the recognition of the textual directives is very unreliable and the metronome directives are not recognized at all. We, therefore, assume in the following that no information about tempo is retrieved from the sheet music data. However, if, for a given score track, the estimated default bar sequence  $\delta^*$  is available, a mean tempo can be computed using the duration  $T_{\text{audio}}$  of a corresponding audio recording, and the estimated number of bars of the score track  $D^*$  as will be discussed in Section 5.5. Otherwise the tempo has to be guessed or set to a fixed value. Even though the algorithms for calculating alignment paths or matches as described in Chapter 3 are designed for and capable of handling differences in tempo, it turns out that for large differences in tempo, of factors of 3 or higher, these methods start to become less reliable [45].

The above discussion shows that the quality of the OMR results is a crucial factor for the success of our approaches. Missing and unreliable information in the OMR results constitute



a main difficulty for this project. In Chapter 5 we discuss how some of the less serious errors can be detected and even corrected in a postprocessing step by considering inconsistencies in the context of the OMR output. For the case of more severe errors, an automatic detection and correction through postprocessing based on the available information seems not practical anymore. Instead, one would have to make improvements to the OMR process itself, which lies outside the scope of both this thesis and the PROBADO project. To offer a solution to the shortcomings of the OMR results that are critical for our approaches, Chapter 5, therefore, presents methods for the streamlined manual correction of critical aspects of the OMR results.

### 4.3 Framework and Workflow

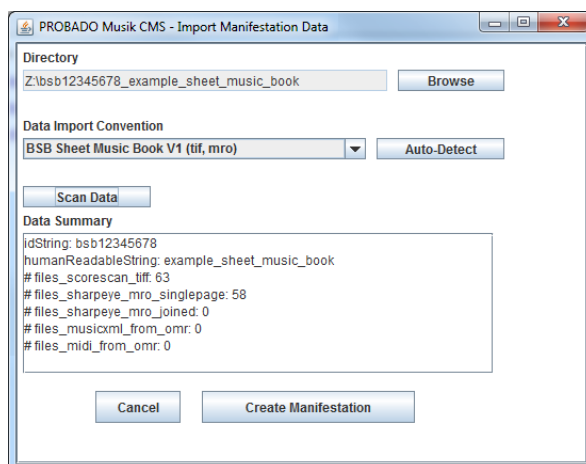
In this section, we describe a concept for a document organization system (DOS) that allows adding content to and managing content in a PROBADO music repository by means of a graphical user interface. This tool combines mechanisms for the automatic organization with graphical interfaces for manual inspection, editing, and correcting. Since libraries follow high standards in data quality and users expect the content provided by libraries to be checked for errors, it is very important to give the library operators the opportunity to manually correct mistakes that happen in the automatic processing. Since music documents often comprise exceptions and peculiarities that can cause automatic approaches to fail, mechanisms for streamlined editing and correction of mistakes are indispensable.

We explain the concept by giving an example of how the system would be used by a library employee, which we will also refer to as the *user*. Let us consider a fictitious dataset including the real-world challenges described earlier in this chapter. The dataset contains one parent-level work that, according to a work catalog that is used as a reference, is divided into 4 leaf-level works.

```
work:
+ Piece 1
  + Movement 1
  + Movement 2
  + Movement 3
  + Movement 4
```

We assume that there are two manifestations given. One is a sheet music book, the other is an audio CD. Both manifestations contain all four movements of the *Piece 1*. However, on the CD, the division of the piece into CD tracks is different from our reference division. Here is how the tracks on the CD are organized:

```
CD 1:
+ Track 1: Movement 1
+ Track 2: Movement 2 (first part)
+ Track 3: Movement 2 (second part)
+ Track 4: Movement 3 and Movement 4
```



**Figure 4.7.** Input mask of the PROBADO Document Organization System for creating a new manifestation based on a given directory containing the relevant files.

On the CD, the second movement is split across two CD tracks. Movements 3 and 4 are merged to a single CD track. Now, let us consider the library employee whose task is to build up the repository from the given data. Starting with an empty database for the metadata, the user adds 5 work entries to the database. One entry represents the whole “Piece 1” while the other four entries represent the individual movements. As the next step, the user wants to add the two manifestations (one audio CD and one sheet music book) to the repository. Each manifestation is given as a single directory containing all the files that belong to the manifestation. To add the two manifestations to the repository, the user performs the following steps.

- Load up the DOS and select “Add new manifestation” as a task.
- The DOS asks the user to select a directory that contains the files of the manifestation to add, see Figure 4.7. The user selects the directory containing the scans and the OMR results for the sheet music book. Based on the directory name and the names and types of files included in the directory, the DOS automatically recognizes that the selected manifestation contains a sheet music book with known conventions and file formats. It outputs a summary of the recognized manifestation, e.g., the amount of scanned pages and OMR result files, for the user to check. The user proceeds by pressing a button to confirm the creation of a new manifestation entry in the repository from the given data.
- The user is presented an input mask that is divided into three main parts. The top part, which is depicted in Figure 4.8 asks the user to enter some global information about the manifestation such as a title, creation date of the data, and contributors, e.g., the institution that performed the digitization of the sheet music. The center part, depicted in Figure 4.9, allows the user to choose some constraints about the composers and works relevant for the manifestation. These constraints narrow down the works considered in the subsequent step of track identification. This mechanism is useful for both manual editing as well as automatic processing especially for large repositories containing many thousands of works.

Information for Manifestation

title	example_sheet_music_book		
publication_date	2009		
identifier	12345678		
contributor	Example Institution	create new	
role	digitizer	create new	delete
add contributor			
apply changes		discard changes	

**Figure 4.8.** Input mask for the basic information of a manifestation such as title and publication date. In the lower part of the mask, persons or institutions can be declared as having a particular role in contributing to the creation of the manifestation.

Global Constraints

work_contributors	remove	add
Example Composer 1		
works	remove	add
Example Composer 1, op.1, [1] Piece 1		
apply changes		discard changes

**Figure 4.9.** Two types of constraints can be used to narrow down the selectable works in the subsequent step of track identification. If one or more “work\_contributors” (which usually are persons in the role of the composer) are added to the upper constraint list, only works with participation of these contributors are considered in the subsequent steps. Using the second constraint list, the selectable works can be further narrowed down by only allowing a particular set of works. Note, that this may include entries for parent-level works such as complete operas, sonatas, or song cycles. When adding an entry for such a parent-level work, all children of that work in the hierarchy will be enabled for selection in the subsequent steps.

## Content (Expressions contained in this Manifestation)

first filename: **bsb00003513\_00001.tif**  
total files: **23**    
total expressions: **4**

1.        mark checked

From    To     
page line bar page line bar

**SONATE**  
Joseph Haydn  
Allegro

Work (leaf-level):   
**Example Composer 1, op.1, [1] Piece 1 - [2] Movement 1**

reference expression: **-none-**

Expression:   
**[2][sheet music], Example Publisher 1 (publisher), 2009**

Expression Information

realized\_date

contributor   role

2.       mark checked

From    To     
page line bar page line bar

Example Composer 1, op.1, [1] Piece 1 - [3] Movement 2

3.       mark checked

From    To     
page line bar page line bar

Example Composer 1, op.1, [1] Piece 1 - [4] Movement 3

4.       mark checked

From    To     
page line bar page line bar

Example Composer 1, op.1, [1] Piece 1 - [5] Movement 4

recognized title(s):

**Figure 4.10.** Input mask for editing data regarding the content of a sheet music manifestation. The content is segmented into score tracks with each score track being represented as a blue box inside the content area. A sheet image browser interface can be used to display the start and end location of each score track inside the sheet music book. It can also be used to select locations inside the book by visually browsing through the pages of the book and selecting a target bar with the mouse pointer.

The bottom part represents the actual content of the manifestation, see Figure 4.10. At the top of this section labeled “content”, a short summary of the content is given. The task that needs to be completed at this part is the track segmentation and identification. More specifically, the content of the manifestation is to be divided into segments, and each segment is to be identified as one of the leaf-level works specified in the metadata database. In the case of a sheet music manifestation, as depicted in the figure, these segments represent what we refer to as score tracks. In the optimum case, the user would simply run a function “auto-calculate all” from the program menu causing the DOS to correctly segment and identify the four score tracks of our example data set as shown in the figure. Details on how such an automated detection can be performed computationally are given in Chapter 6. In the user interface, each score track is represented by a blue box inside the content section. In Figure 4.10, four of these boxes are shown. Here, the first box displays all information available for that score track and the last three boxes are set to “minimized”, which means that only the basic information is shown. The most important information contained in each box is the start and end location of the corresponding score track as well as the leaf-level work entry it is identified with. For example, the start and end location of the first track are set to 8\_1\_1 and 12\_6\_7, and it is identified with the leaf-level work entry specified as “Example Composer 1, op.1, [1] Piece 1 - [2] Movement 1” (the values in brackets are internal IDs of the database tables and can be ignored in the following). In addition to this basic information, the boxes contain a row of buttons that offer functionality such as applying and discarding changes, minimizing or maximizing the box, merging or splitting tracks, as well as marking a track as “checked”. The maximized box for the first track shows additional user interfaces that assist in verifying and editing the start and end location. A sheet image browser visually presents the start end location inside the scanned sheets and can also be used to conveniently set locations by browsing to the correct image file and clicking the target bar with the mouse pointer. On the right side of the box, a list of text elements recognized in the sheet music scans is displayed. It poses another mechanism to aid the verification and selection of regions in the sheet music book. The text elements are sorted by the score book location they were found at, i.e., page, grand staff, and bar number. This sorted list of text elements is linked to the input fields for the start and end location of the leaf-level score track. Selecting a range of entries in that list sets the region of the score track to start at the location of the first selected entry and to end at the location of the last selected entry. At the bottom of the maximized box for the first track, additional input elements are shown that concern an “expression” entry for that track. According to the metadata scheme introduced in Section 4.1, each leaf-level track that is part of a manifestation must correspond to an expression entry. If no expression entry for a newly created leaf-level track exists, the interface offers the user to create suitable entries using information that can be set in the lower part of the content box for the leaf-level track. This information includes a creation date and a set of contributors. In the case of sheet music, the creation date refers to the year when the transcription was created, and a contributor may for example be the editor of the transcription.

In the metadata scheme, each leaf-level work is assigned a so-called reference expression and each expression is assigned a so-called *reference manifestation*. This mechanism is useful to link a work entry to corresponding reference content in a manifestation (sheet

music book or audio CD collection). Without reference content, all that is known about a work is metadata such as title and composer. Having reference content available for a given work allows us to uniquely specify locations and regions in that work. This is, for example, useful for handling special cases such as missing sections or cadenzas as will be discussed in Section 4.4. Furthermore, having reference content available for each work allows us to perform content-based comparisons to decide to which work a given content-based query belongs to. This type of content-based comparison can be useful for the task of track identification, see Chapter 6. In our example case, the sheet music manifestation and the expressions that are added to the repository by the user are the first entries of this type in the database and, therefore, are automatically used as the reference. Having checked and verified the correctness of the score tracks, the user instructs the DOS to apply and saves all changes.

- The user tells the DOS to add another manifestation.
- This time, the user selects the directory holding the files of the audio CD manifestation. Besides the audio files, this folder contains a textfile with titles for each CD track.
- The user is presented an input mask that looks almost identical to the sheet music case, see Figure 4.11. A main difference is that the leaf-level tracks created in the content part are audio tracks instead of score tracks. For this reason, the format of the start and end location of each track looks differently. Instead of a virtual book that can be used to display and edit location markers, here, a simple audio player is presented. Instead of a list of recognized text elements, the list that is linked to the fields for start and end location shows a list of titles of CD tracks. All other components of the input mask are the same as in the sheet music case.

In our example case, the division of *Piece 1* into CD tracks is not the same as the audio tracks that need to be created to fit the reference division specified in the metadata database. CD track 1 corresponds to the first movement leading to an audio track with boundaries 1.1 to 1.1. The second movement is split across CD tracks 2 and 3. Therefore, the corresponding boundaries for the audio track are 1.2 to 1.3. This way, both CD tracks are virtually merged into a single audio track that conforms to our reference division specified in the metadata. CD track 4 contains two movements. Let us assume that the time of transition between *Movement 3* and *Movement 4* in the CD track is 04:32.19. Two audio tracks are created. The first one represents *Movement 3* and has its boundaries set as 1.4 to 1.4\_04:32.18. This specifies the region containing CD track 4 from the beginning to just before time position 04:32.19. The second one represents *Movement 4* using the boundaries 1.4\_04:32.19 to 1.4, which specifies the region containing CD track 4 from 04:32.19 to the end. As can be seen from the above example, the content can be freely partitioned into contiguous blocks regardless of how it is divided into disks and CD tracks. The same principle holds for the case of sheet music books.

Having checked and verified the correctness of the leaf-level audio tracks, the user instructs the DOS to apply and saves all changes.

- The user has finished adding the two manifestations. Now, the DOS schedules and performs the next steps such as synchronization between the leaf-level tracks.

Content (Expressions contained in this Manifestation)

first filename: **bsbmab007063693\_cd001\_001.wav**  
total files: **4**              
total expressions: **4**

---

1.                               mark checked

From   To    
disctrack time      disctrack time

cd track title(s):

---

Work (leaf-level):    
reference expression:

Expression:

Expression Information     

realized\_date

contributor   role

---

2.                           mark checked

From    To     
disctrack time      disctrack time      Example Composer 1, op.1, [1] Piece 1 - [3] Movement 2

---

3.                           mark checked

From    To      
disctrack time      disctrack time      Example Composer 1, op.1, [1] Piece 1 - [4] Movement 3

---

4.                           mark checked

From     To     
disctrack time      disctrack time      Example Composer 1, op.1, [1] Piece 1 - [5] Movement 4

**Figure 4.11.** Input mask for editing data regarding the content of a audio CD manifestation. The content is segmented into audio tracks with each audio track being represented as a blue box inside the content area. A simple audio player can be used to listen to the audio content near the start and end location of each audio track inside the CD collection.

The same interfaces and concepts that are used to add manifestations to the repository, can also be used to edit and make corrections to manifestations that are already part of the repository.

## 4.4 Handling Special Cases

Using the document organization system described in the previous section, it is possible to conveniently perform the segmentation and identification of sheet music books and audio CD collections according to the reference set of leaf-level works stored in the metadata database. However, it is not guaranteed that all the resulting leaf-level tracks contain exactly the same musical content. There are special cases that can cause certain sections of content to be missing or to be different when being compared to other tracks. Such special cases need to be handled for a successful synchronization. In this section, we discuss several kinds of special cases relevant for our scenario, namely cadenzas, optional sections, missing sections, and alternative sections. We also refer to these special cases as *COMA* (Cadenza, Optional, Missing, Alternative) and denote the four possible cases as *COMA classes*.

As already pointed out in the previous section, each leaf-level work entry is assigned reference content by means of a reference expression which, in turn, has a reference manifestation. The COMA special cases are assigned to individual expressions, because the expression entity represents musical content independently of the particular print or audio CD collection the content is published on. For each expression, a list of COMA cases is maintained. We refer to entries in that list as *COMA entries*. Each COMA entry specifies a COMA class, a corresponding region in the reference manifestation of the expression, as well as additional information depending on the COMA class. Figure 4.12 shows some example COMA entries as they might look like in the document organization system. In the following, we will discuss each of the four COMA classes in more detail.

- **cadenza:** Cadenzas are sections performed by soloists without accompaniment. Usually, the soloists are free to choose what to sing or play themselves. In some cases, cadenzas are proposed and written out by the composer of the piece including the cadenza itself. In other cases, other composers have written cadenzas for pieces they did not compose themselves. Performers can even choose to perform their own cadenza or to improvise freely. In sheet music books, cadenzas do not have to be included in the score. Instead, one often finds a bar with a rest and a fermata with a textual hint that a cadenza is to be performed at this position, see Figure 4.13 for an example. In audio recordings, however, the performance of cadenzas is mandatory.

Since cadenzas are not necessarily part of the original work in which they are embedded, cadenzas may be assigned special work entries of their own. Besides offering the advantage of storing additional information on the cadenza, such as a composer or date of creation, this way, the document organization system can keep track of which cadenza is written or played in which expression. In the example depicted in Figure 4.12, a section ranging from 53.2.1 to 53.5.6 is marked as a cadenza. The cadenza is identified with a separate work entry that specifies **Example Composer 2** as the composer and ‘‘Cadenza Name’’ as the title of the cadenza.



Cadenzas, Optional, Missing, or Alternative parts (COMA):

1.	class <input type="text" value="cadenza"/>	From <input type="text" value="53_2_1"/>	To <input type="text" value="53_5_6"/>	Work (leaf-level): <input type="text" value="Example Composer 2, 'Cadenza Name'"/>	<a href="#">create new</a> <a href="#">delete COMA</a>
2.	class <input type="text" value="optional"/>	From <input type="text" value="54_2_1"/>	To <input type="text" value="54_2_8"/>		<a href="#">delete COMA</a>
3.	class <input type="text" value="missing"/>	Location <input type="text" value="before"/>	<input type="text" value="54_3_4"/>	Region of Missing Section in Reference Content From <input type="text" value="44_1_1"/>	To <input type="text" value="45_1_13"/> <a href="#">delete COMA</a>
4.	class <input type="text" value="alternative"/>	From <input type="text" value="59_1_1"/>	To <input type="text" value="60_2_9"/>	Alternative Group ID String: <input type="text" value="group A"/>	<a href="#">delete COMA</a>
5.	class <input type="text" value="alternative"/>	From <input type="text" value="61_1_1"/>	To <input type="text" value="64_2_12"/>	Alternative Group ID String: <input type="text" value="group A"/>	<a href="#">delete COMA</a>

[add new COMA](#)

**Figure 4.12.** Illustration of a concept for the input mask for COMA entries in the document organization system. The Figure shows example COMA entries for the four different COMA classes *cadenza*, *optional*, *missing*, and *alternative*.

- optional:** A section in a given expression about which is known that it may be skipped in other expressions of the same work can be marked optional. This information can prove very useful when synchronizing to an expression where this section is omitted, but is not declared missing. Besides the region boundaries, no additional information is needed for the COMA entry. In the example depicted in Figure 4.12, a section comprising 8 bars ranging from 54.2.1 to 54.2.8 is marked optional.
- missing:** If, in a given expression, a section is missing that is contained in the reference expression of the corresponding leaf-level work, this section has to be declared missing by a COMA entry of this class. For this COMA class, a location is specified instead of a region. The location marks the point where the missing section would have to be inserted to match the reference expression. As additional information, the region boundaries of the missing section in the reference content of the leaf-level work can be given to exactly specify which content is missing. In the example depicted in Figure 4.12, a region in the reference content ranging from 44.1.1 to 45.1.13 is missing in the current expression. It would have to be inserted right before the location 54.3.4 in the reference manifestation of the current expression to match the reference content.

Note that entries of type “optional” and “missing” are related in a way that sections that are declared optional in a reference expression can be declared missing in non-reference expressions of the same leaf-level work. But unlike entries of class “missing”, entries of class “optional” are not obligatory, which can lead to cases where a part is declared missing even though it is not declared optional in the reference expression.

- alternative:** This COMA class is usually used only for sheet music expressions. In the sheet music, there may be sections for which two or more alternatives are given to choose from. See Figure 4.14 for an example. Each alternative in a group of interchangeable alternative sections must be declared by a COMA entry. For all of these entries, the same unique identifier string is set in the additional information so the document organization

The image shows a musical score for Mozart's Violin Concerto in G major, KV 216, Adagio. The score is divided into three systems. The first system contains the piano part (treble and bass clefs) and the violin part (treble clef). The piano part starts at bar 42 with a *p* dynamic and a *crescendo* leading to a *f* dynamic at bar 44. The violin part also starts at bar 42 with a *p* dynamic and a *crescendo* leading to a *f* dynamic at bar 44. The violin part has a trill (tr) at bar 44. The second system contains the orchestra part (treble and bass clefs). The orchestra part starts at bar 42 with a *p* dynamic and a *crescendo* leading to a *f* dynamic at bar 44. The orchestra part has a *coll' arco* instruction at bar 44. The third system contains the piano part (treble and bass clefs) and the violin part (treble clef). The piano part starts at bar 44 with a *f* dynamic and a *crescendo* leading to a *p* dynamic at bar 46. The violin part starts at bar 44 with a *f* dynamic and a *crescendo* leading to a *p* dynamic at bar 46. The violin part has a *simile* instruction at bar 44. The score is marked 'TUTTI' at bar 44.

⇒ T. 44, Violino principale: Hier ist eine Kadenz zu spielen.

**Figure 4.13.** Example for a cadenza indicated in a score for Mozart’s Violin Concerto in G major, KV 216, Adagio. The cadenza is to be played at bar 44, where all instrument groups except for the solo violin are given a rest with a fermata. A footnote linked to the solo violin part at bar 44 remarks that a cadenza is to be played. The notes that are written in bar 44 of the solo violin part are not to be played together with the remaining orchestral parts. Instead, they indicate how the cadenza, which does not start before reaching the general pause, is expected to end and to lead back to the orchestral part.

system can tell which alternatives are mutually interchangeable. In the example depicted in Figure 4.12, two alternatives are assigned to the same identifier string group A. The two entries express that in a performance of the sheet music one has to expect either section 59\_1.1 to 60\_2.9 being performed or section 61\_1.1 to 64\_2.12 being performed instead.

The COMA entries are an important piece of information for synchronizing leaf-level tracks. This is true not only for sheet music-audio synchronization but also for audio-audio synchronization. Note that in the sheet music-audio case, it are COMA cases that can lead to performance sequences  $\pi$  that contain elements  $\uparrow$  standing for content that is not part of the sheet music. In practice, one would like to be able to automatically detect all COMA cases and generate the correct COMA entries. However, depending on the COMA class and the data types (sheet music-audio, audio-audio), the difficulty of this task can range from feasible to very hard. For example, if a bar with a keyword “cadenza” is found in a leaf-level score track, it seems feasible to correctly detect the cadenza in a leaf-level audio track that is to

Poco Andante, ma sempre Alla breve. Zz ⊕

Bei Weglassung des Chores sind die hier folgenden zehn Schlusstakte unmittelbar anzuknüpfen.  
 If the chorus be left out, the following ten final bars should be immediately connected with the foregoing.  
 Si l'on supprime le chœur, on enchaînera immédiatement les dix mesures finales qui suivent.  
 Ha a kórust elhagyjuk, akkor közvetlenül az itt következő 10 záróütemre térünk át.

Poco Andante, ma sempre Alla breve. Zz ⊕

F. L. 14.

**Figure 4.14.** Example for a case where alternative options are given in a sheet music track. The example shows the Breitkopf & Härtel edition of “A Faust Symphony” by Franz Liszt near the end of the third part (Mephistopheles). At the end of the grand staff depicted in the figure, a textual note explains that one has to choose to either just continue with the final ten bars in order of notation, or to follow the jump marker ⊕, to jump to an alternative ending part that involves the performance of a choir.

be synchronized with. A suggestion for an approach to solving this task will be given in Chapter 7. However, if it is not known where a cadenza (or a different COMA case) occurs in a leaf-level track and if it even occurs at all, the task becomes much harder because one has to expect it to happen at any position. Because, at this point, a reliable automatic detection of all but the most simple COMA cases is not available, we will assume that the COMA entries are entered manually by the user. The automatic detection of COMA cases poses an interesting and challenging topic for future work.



## Chapter 5

# From Sheet Music To Note Events

An important step for the content-based comparison of sheet music data and audio data is the generation of note events specified by absolute pitches, onset times, and durations (see Figure 2.2) from the sheet music data. This task requires several steps. First, the sheet music is converted to symbolic score data through optical music recognition (OMR). The symbolic score data we obtain from the OMR results basically consists of a notation bar sequence  $\nu$  (see Figure 3.2) with each bar  $\nu_b \in \nu$  containing a set of symbols, e.g., clefs, key signatures, notes, and rests. Additionally, for each page, there is a set of global symbols that are not (yet) attached to any single bar in particular, e.g., textual directives, slurs, and dynamics symbols. The task, now, is to correctly derive note events specified by onset times, pitches and durations from the symbolic score data. Presuming that the symbolic score data is complete and accurate with respect to all the symbols that affect these parameters, this task can be solved by applying the basic rules of music notation. However, in practice, not all the relevant information is output by the OMR process, and some part of the output information is inaccurate or unreliable. In this chapter, we discuss strategies for both dealing with the most critical shortcomings of the OMR output in a streamlined way and for making best use of the available data in our scenario. In Section 5.1, we discuss the task of optical music recognition and point out the most critical shortcomings for our project. In Sections 5.2 and 5.3 we present approaches to encounter some of these shortcomings via postprocessing techniques. To actually derive note events specified by onset times, durations, and pitches, one important task is to determine the sequence in which the bars are to be played, i.e., the default bar sequence  $\delta$ . The task of deriving an estimate  $\delta^*$  by detecting repeats and jumps in the score is discussed in Section 5.4. A robust method for deriving onset times is presented in Section 5.5. The principles of deriving pitches by applying the basic rules of classical Western music notation are outlined in Section 5.6. Finally, in Section 5.7, we point out certain difficulties that can occur in orchestral scores and propose a streamlined method to manually encounter the aspects that are most critical for our application.

## 5.1 Optical Music Recognition

As already pointed out in Section 2.1, the term optical music recognition (OMR) is commonly used to refer to transformations from the sheet music domain to the symbolic domain. Actually, there is a wide spectrum of transformations depending on the kind of input and output data. For example, starting with a vector graphics image of a score, the task of assigning labels for musical entities such as `note head`, `stem`, `treble clef`, `staff line` to each of the shapes could already be considered as OMR. However, OMR usually covers a lot more than that. For example, when starting with a bitmap image, one crucial subtask of OMR consists in grouping the pixels to musically meaningful shapes and relating these shapes to musical entities.

OMR systems can differ in how far and deep they try to reach into the symbolic realm with their transformation. Many OMR systems stop after having identified most of the shapes as musical symbols and having interpreted their basic meaning and relations. Shapes that are not recognized are often ignored. Furthermore, higher level semantics are often neglected. For example, an OMR system might recognize repeat signs, but not necessarily does it also interpret their musical meaning, i.e., the effect on the sequence of bars that is to be followed when playing the piece. As another example, consider text-based information contained in sheet music. Most OMR systems are able to recognize text and even to distinguish between song lyrics and other text. However, the systems usually do not further distinguish other possible functions of the text elements, such as title heading, section name, tempo directive, jump directive (da capo, fine, etc.), or instrument name.

Several commercial and non-commercial OMR software systems exist.<sup>1</sup> Three of the more popular commercial systems that operate on common Western classical music are SharpEye<sup>2</sup>, SmartScore<sup>3</sup>, and PhotoScore<sup>4</sup>. Two of the more prominent examples of free OMR systems are Gamera<sup>5</sup> and Audiveris<sup>6</sup>. However, Gamera is actually a more general tool for document image analysis that requires to perform training on the data to be recognized [39]. An OMR module for recognizing classical Western music in Gamera called AOMR2 has been developed and used in the literature [38, 37]. However, it seems to be no longer under development or available for download. Instead, there are OMR modules specialized on lute tablature and Psaltic neume notation. Audiveris is an open-source OMR software written in Java, but, currently, is not competitive in terms of recognition rates compared to the commercial products.

Evaluating and comparing the general performance of OMR systems is a non-trivial task. In particular, it is not clear how the performance is to be measured [37]. A comparison of some aspects of OMR systems mentioned above and a detailed discussion of the problems regarding evaluation of OMR can be found in the work by Byrd et al. [18, 17]. In general,

---

<sup>1</sup>Online lists can be found at <http://www.informatics.indiana.edu/donbyrd/OMRSystemsTable.html> and <http://www.music-notation.info/en/compmus/omr.html>

<sup>2</sup><http://www.visiv.co.uk>, November 2009

<sup>3</sup><http://www.musitek.com>, November 2009

<sup>4</sup><http://www.neuratron.com>, November 2009

<sup>5</sup><http://gamera.informatik.hsnr.de>, November 2009

<sup>6</sup><https://audiveris.dev.java.net>, November 2009

the quality of OMR results strongly depends on the quality of the input image data and the complexity of the underlying scores. More complex scores tend to result in more OMR extraction errors. Here, typical OMR errors are non-recognized symbols, missing accidentals (also in key signatures), missing beams, missing or extra barlines, erroneous time signatures, and multi-staff grand staves that are mistakenly split apart horizontally into smaller groups or individual staves.

A general observation is that most commercially distributed OMR systems are designed to help users to create a symbolic representation from a printed and scanned score so it can be edited, rearranged, reprinted, or archived. In that case, the main purpose of the output symbolic representation is still to serve as an image that is to be read and interpreted by human readers. Therefore, the recognition of higher-level semantics by the software such as tempo, repeats, jumps (*da capo*, alternative endings), or link-ups of staves in consecutive multi-staff systems, is not important. More important, in this use case, is that the musical symbols are correctly recognized. Since recognition errors are expected to occur, users are expected to correct these errors manually. The OMR systems provide editors and tools that assist in this task.

The demands on OMR in the use case of our digital music library project are very much different from the above. Since the symbolic score data created through OMR is only used in the process of organizing the content and calculating the synchronizations, but is never directly presented to the user, we do not require the OMR results to be perfectly accurate, or to even “look” right. We only need it to be accurate enough to allow our approaches to automatic organization and synchronization of the data to work. For our scenario, a thorough manual correction of the OMR results is not an option, because the amount of sheet music is simply too large. As of October 2009, there have been 50,000 pages of sheet music in 292 sheet music books available for the project. Considering, that the number of printed sheet music books available at the Bavarian State Library in Munich exceeds 100,000, it becomes clear that a manual correction of each page is not practical.

There are two more requirements on OMR software to be useful for our scenario. For the application of linking bars that are visible on sheet music scans presented to the user to temporal regions in corresponding audio recordings, we need the OMR output to contain accurate information on the position and size of each bar in the sheet music images. Finally, the software should provide a simple way of being integrated in a fully automatic processing chain. Unfortunately, the bigger commercially distributed OMR programs are bound to graphical user interfaces that require human interaction to perform the recognition. The OMR module of the SharpEye2 Music Reader used in our project, see Section 2.3, can be run as a command-line application, which makes it easy to integrate it into the automated workflow.

Our approaches for the automatic organization and synchronization operate on two kinds of data extracted from sheet music books. The first kind of data is structural information such as title headings and indentations of grand staves that is useful for the track segmentation and identification of sheet music books, see Chapter 6. Because commercially distributed OMR software is intended to be used with individual pieces and movement, but not with complete sheet music books, recognition of titles and boundaries for the pieces comprised in the books is not provided. The second kind of data is musical content, which, in our case, we consider to

be the note events derived from the scores. Since our content-based algorithms work on these note events, we need to extract note events from the OMR output as accurately as possible. We discuss strategies for extracting such information from OMR results in the subsequent sections of this chapter.

Printed sheet music usually follows a rich set of rules, regarding aspects of engraving, notation, and layout. Furthermore, there are high-level rules that dictate which constellations of symbols are valid. Because many of these rules are not incorporated into current OMR software, they can be used to correct errors in the OMR output data through postprocessing. Unfortunately, the set of valid rules can differ significantly between different types of scores, music, and instrumentation. Special types of notation and exceptions are specific to particular types of music. To be able to make use of the rules that apply to the given material, one first has to decide which rules actually can be applied. Assuming the type of music or instrumentation is known, OMR postprocessing can be performed to fix some of the recognition errors. We will discuss some pragmatic approaches for that in the following section.

## 5.2 Postprocessing OMR Data Using Simple Heuristics

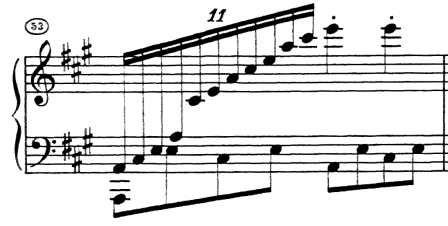
It seems clear that there still has to be put a lot of effort into the development of OMR software until all symbols and semantics of classical Western sheet music can be extracted reliably. Our main goal is to get as much and as accurate information out of the results of the OMR software used in this project as is needed to make automatic organization and synchronization work in a digital library environment. Since the OMR engine used in this software does not seem to make use of higher-level or instrumentation-specific rules, we can detect and correct many of the errors in the OMR output using pragmatic ad-hoc approaches in a postprocessing step.

Several ad-hoc approaches for detecting and correcting OMR errors as well as inferring important semantic aspects have been developed and implemented. However, as pointed out in the previous section, the set of rules that can be expected to apply for the sheet music depends on several factors such as instrumentation. Since in the early phase of the PROBADO project the music collection consisted of piano music only, some of the rules used in these approaches are specific to piano music. Even though some of these rules may apply more generally, in the remainder of this section, we assume that the underlying material is piano music.

Grand staves in scores for piano music usually consist of two staves. The upper staff is played with the right hand, and the lower staff is played with the left hand. However, there are some exceptions to this. Unlike scores for ensembles where grand staves are simply groups of individual staves that each form a coordinate system for time and pitch, the two staves for the left and right hand can be treated as a single coordinate system as shown in Figure 5.1. Here, a note group that is connected through a single beam is spread across the two staves. This mechanism is often used to avoid notes with many auxiliary lines between the two staves.

In the following, we discuss a set of modules that detect and correct errors and derive semantics from OMR results of piano music based on simple heuristics. Each of these modules can be





**Figure 5.1.** Example for a beamed note group that stretches over two staves in piano music.

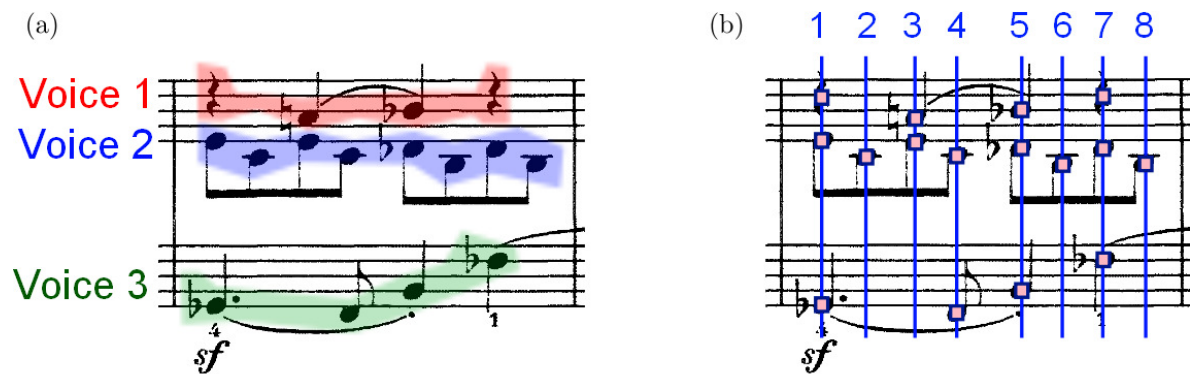
applied to the data individually. Even though these modules are rather simple and naive, they can eliminate many errors that would otherwise have a negative influence on the content-based comparison of note events and audio recordings.

### 5.2.1 Voice Detection

Our starting point is the OMR data output by the SharpEye software. For each bar, we consider symbols representing chords, notes, and rests. In the following, we will use the term *chord* to refer to both chords and individual notes, i.e., we consider individual notes as a special case of a chord containing just a single note. Let us use the term *voice* to refer to a sequence of chords and rests that are played consecutively and without a gap. In piano music, several voices that are to be played simultaneously can coexist on the two staff system. An example bar of piano music with three voices is depicted in Figure 5.2(a). To allow a reader of the score to better differentiate between individual voices, music notation usually follows certain conventions.

- (VC1) Events that are meant to occur at the same time share the same horizontal position.
- (VC2) Each chord or rest belongs to one voice only.
- (VC3) For two voices written in a single staff, the stems of the upper voice point upwards, and the stems of the lower voice point downwards.
- (VC4) Each voice individually fills out the duration of the complete measure with either rests or notes.
- (VC5) Voices usually do not intersect or change position relative to each other.

In addition to these conventions, a common visual aid for the reader is that consecutive events that belong to the same voice can often be identified by being grouped by beams, slurs, or ties. In the cases where the above conventions hold, one can use a simple algorithm to determine the voices of a given bar of sheet music. Let us consider the example depicted in Figure 5.2. The algorithm is divided into two steps. In the first step, the amount of voices present in the bar is estimated by making use of (VC1) and (VC2). To this end, we group chords and rests that roughly share the same horizontal position into so-called *raster groups*. In Figure 5.2(b), we see that for our example bar we get 8 of these raster groups. Then, we assume the



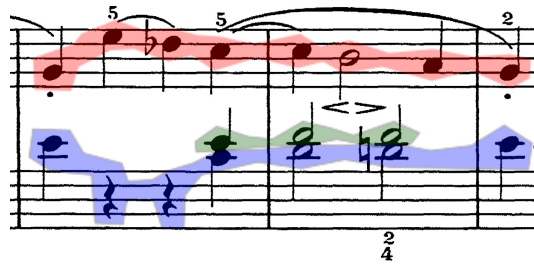
**Figure 5.2.** Example for a bar of piano music with three voices. (a) The notes and rests of each voice are highlighted with different color. (b) The amount of voices in the bar is estimated by first grouping all chords and rests with the same horizontal position and then counting the maximum group size.

amount of voices that is present in the bar to be the maximum number of chords/rests found in any of these groups. In our example, this maximum number is 3, and it is found in the first, fifth, and seventh group. The second step is to map each chord/rest to one of the voices. We start with the groups where the maximum number of voices is present. Here, we assign voice numbers by vertical position. The upper chord/rest is assigned to voice 1, the middle chord is assigned to voice 2 and the lower chord is assigned to voice 3. Now the chords/rests of the remaining groups need to be assigned. This is done by using the above conventions for stem direction and non-intersection of voices that we assumed to hold and can be assisted by considering beams, slurs, and ties.

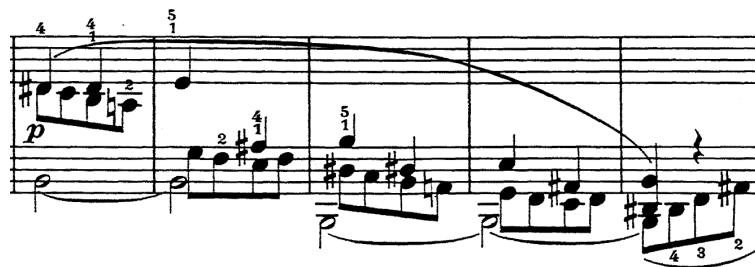
This rather crude and simple algorithm already works well in most cases, but certainly not in all cases. In situations where following the above conventions (VC1) to (VC5) would lead to overlapping symbols or degraded readability of the notation, these conventions are often suspended. Examples for such cases are depicted in Figures 5.3 through 5.5. Even though there may be again conventions on how such cases are usually handled, in general, the engraver may prefer custom solutions for each particular case that disrespect conventions in favor of improved readability. This makes finding an algorithm that always finds the correct voices very hard. In many cases, the voices can only be inferred by deciding which interpretation of the notation makes most sense in the current context. This decision requires not only pondering existing conventions and rules on several semantic levels simultaneously, but also the anticipation of intended meaning aside from the known rules of music notation. Some curious examples for such exceptional cases of music notation have been collected by Byrd<sup>7</sup> [14].

Incorporating more conventions and indicators, e.g., note durations, into an algorithm for voice detection may lead to improved results compared to our simple approach described above. We will discuss a more general approach of incorporating conventions and indicators into postprocessing for correcting errors and deriving semantics in Section 5.3. However, considering that in our real-world scenario, the data must additionally be expected to suffer from OMR errors such as missing or additional chords and rests as well as errors in note

<sup>7</sup><http://www.informatics.indiana.edu/donbyrd/InterestingMusicNotation.html>



**Figure 5.3.** Example for a voice appearing within a bar of piano music. Even though in the first bar of this example at the 4-th quarter note there are three voices, the middle voice is not preceded by rests to fill up the complete measure.



**Figure 5.4.** Example for having three voices written in one staff in piano music. To avoid overlapping symbols, the notes that coincide in onset time are no longer aligned horizontally.

duration, in general, cases where the voices cannot be identified unambiguously cannot be avoided.

### 5.2.2 Key Signature Correction

In piano music, key signatures are expected to be the same for the left hand and right hand staff at any time. Therefore, different key signatures in the left-hand and right-hand staff of OMR results are typically caused by OMR errors, see Figure 5.6. In case of the SharpEye OMR engine, it is much more likely that accidentals are missed out rather than extra accidentals being added. Therefore, a good strategy to handle different key signatures in the left and right hand is to choose the one which has more accidentals. Since key signatures possibly affect the pitch of many notes throughout a whole staff of music, fixing these obvious errors can significantly improve the accuracy of note events extracted from the sheet music with little effort. Note, however, that this simple algorithm does not detect cases where staff signatures in both staves have the same amount of missing accidentals.

In the sheet music books “Beethoven Piano Sonatas Volume 1 & 2” by G. Henle, disagreeing key signatures were found in 258 of a total of 3693 grand staves, which is a rate of roughly 7%.



**Figure 5.5.** Example for a rather complex situation that comprises 8 voices in a single bar. However, in cases like this where most of the voices consist of a single tied note event, the technical differentiation between the individual voices is no longer important. The notation may be read as simply suggesting a sequence of eighth notes where each note is held until or beyond the end of the bar. Even though the result would be the same if of the 8 eighth notes, only the 4-th and 8-th one would have been written, the beamed note groups give a nice visual hint on what is the essential part of this bar of music.

### 5.2.3 Time Signature Correction

Similar to the case of key signatures, time signatures in piano music are expected to be the same for the left and right hand at all times. Disagreeing time signatures found in OMR results indicate recognition errors. However, in case of time signatures it is not clear which of the two disagreeing recognition results is the correct one, if any. To deal with this issue, in our simple approach, we calculate the durations of the voices found in the 10 bars that follow the time signature and check if they agree or disagree with the durations implied by the time signatures. Since the detection of voices and the duration of notes and rests is not perfectly reliable, we use a voting mechanism to find the predominant duration among the 10 bars. If one of the time signatures agrees with the predominant voice duration, the other time signature is changed accordingly.

If there is a very clearly predominant voice duration that disagrees with both of the found time signatures, it is likely that both time signatures were recognized incorrectly. In this case, a suitable time signature is generated from the predominant voice duration and the time signatures are replaced. An example is depicted in Figure 5.6.

### 5.2.4 Execution Hint Removal

Another issue for the automatic generation of note events from OMR data of piano music is execution hints that, by the OMR software, are recognized as regular grand staves. Such execution hints are usually found at the bottom of the page being linked to a particular position in the score by some marker, see Figure 5.7. If not handled, this will cause bars and note events found in the execution hint to be appended after the last regular grand staff of the page.

We use a simple heuristic to identify execution hints by searching for single-staff grand staves

**Figure 5.6.** Example for an OMR result for piano music with disagreeing key signatures at the beginning of the grand staff and incorrectly recognized time signatures. The key signatures can be corrected by choosing the one with more accidentals to replace the other. The time signatures can be corrected by finding the predominant voice duration found in the 10 bars following the time signature.

**Figure 5.7.** Example for an execution hint written at the bottom of a page. Execution hints in piano music usually consists of a single staff that is smaller than the regular staves.

that are found below all other grand staves and whose size is smaller than the size of the other grand staves on the page. Grand staves that are classified as execution hints by this heuristic are completely removed from the OMR output.

### 5.2.5 Grandstaff Indentation Detection

In sheet music books that contain several songs or movements, the beginning of a new song or movement is usually indicated by an indented grand staff such as depicted in Figure 5.8. Since the OMR software does not detect the boundaries of individual songs or movements, we use these indentations as indicators for track boundaries, see also Chapter 6.

To detect indentations, we compare the left boundary of all grand staves found on a page.

The figure displays four systems of musical notation. The first system is a grand staff (treble and bass clefs) with a circled measure number '310' at the start. It features piano (*pp*) dynamics. The second system is titled 'Largo appassionato' and includes the markings 'lento sempre' and 'staccato sempre'. The third and fourth systems continue the musical notation with various dynamics and articulations.

**Figure 5.8.** Example for an indented grand staff indicating the beginning of a new movement in the sheet music book “Beethoven Piano Sonatas Vol. 1” by G. Henle.

Assuming, that the smallest left margin found on the page indicates a non-indented grand staff and that margin is roughly the same for all non-indented grand staves on a page, indentations are revealed as significant deviations towards the right. For piano music, the recognition of indentations is pretty stable. However, the resulting segmentation of sheet music books into songs or movements does not always coincide with the segmentation used on audio CD collections. This matter will be further discussed in Chapter 6.

### 5.3 Iterative Approach for Reducing Inconsistency

For the incorporation of many of the higher-level rules of music notation and semantics, the mechanism of considering simple heuristics in isolated modules, as done in the previous subsections, might not be sufficient. The reason for that is that changes made to the OMR output data might affect many rules simultaneously. Finding the correct changes to make may require considering how these changes affect the consistency of the data with possible many different rules across many different semantic levels simultaneously.

In the following, we describe a simple iterative approach for reducing inconsistency of the OMR output data with given sets of rules. Let us use the symbol  $\Omega$  to refer to a given set of symbolic score data obtained via OMR. Rules are specified and collected as elements in a so-called *rule set*  $R$ . Each rule  $r \in R$  provides two basic functionalities. Firstly, given the score data  $\Omega$ , a rule must be able to identify a set of *violations*  $V = V(\Omega | r)$ . Each violation  $v \in V$  is assigned a cost  $c(v | \Omega)$ , which is used as a measure for the inconsistency between the

rule and the data  $\Omega$  caused by the violation  $v$ . We choose the notation  $c(v \mid \Omega)$  to implicate that the violation and its cost depends on the context that is given through the OMR data  $\Omega$ . Secondly, for each violation  $v \in V$ , the corresponding rule must be able to provide a set of candidate modifications  $M_v$  that would eliminate or at least attenuate the violation  $v$  when being applied to  $\Omega$ .

Given a rule set  $R$  and some score data  $\Omega$ , the iterative algorithm makes changes to the data  $\Omega$  trying to minimize the inconsistency of the data with the rule set by performing the following steps:

1. Set the iteration counter  $k := 0$  and initialize the score data  $\Omega^0$  after 0 iterations by setting  $\Omega^0 := \Omega$ .
2. Calculate the violations found in the score data  $\Omega^k$  based on the rule set  $R$  and store them as  $V^k = V(\Omega^k \mid R) = \bigcup_{r \in R} V(\Omega^k \mid r)$ .
3. Calculate the inconsistency measure  $c^k = c(V^k \mid \Omega^k) = \sum_{v \in V^k} c(v \mid \Omega^k)$ .
4. For each violation  $v \in V^k$ , calculate the set of candidate modifications  $M_v^k$  and collect them in the set  $M^k := \bigcup_{v \in V^k} M_v^k$ .
5. For each modification  $m \in M^k$ , apply the modification to the data and obtain  $\Omega_m^k := \text{modify}(\Omega^k, m)$ .
6. For each  $m \in M^k$ , calculate the violations found in the modified score data  $\Omega_m^k$  based on the rule set  $R$  and store them as  $V_m^k = V(\Omega_m^k \mid R)$ .
7. For each  $m \in M^k$ , calculate the inconsistency measure for the modified score data  $c_m^k = c(V_m^k \mid \Omega_m^k)$ .
8. Find the minimum inconsistency after  $k + 1$  iterations  $c^{k+1} := \min_{m \in M^k} c_m^k$ .
9. If there has been an improvement, i.e.,  $c^{k+1} < c^k$ , then set  $\Omega^{k+1} := \Omega_{m_0}^k$  with  $m_0^k = \arg \min_{m \in M^k} c_m^k$ , increase the iteration counter  $k := k + 1$ , and continue with Step 2. Otherwise, terminate the algorithm and output  $\Omega^k$ .

The above algorithm iteratively checks the given data for rule violations, tries out all modifications that are proposed by the violated rules, and then chooses the modification that brings the most improvement with respect to the complete set of rules. If a modification fixes a violation for one rule but causes new violations for other rules, the improvement for fixing the first violation will be diminished or even eliminated by the cost for the newly caused violations. Only modifications that result in an overall improvement concerning all the rules included in the rule set will be performed. A big advantage of this approach is that one no longer needs to worry about how to decide which rules to consult to fix which aspects of the data. One can simply collect all rules and conventions that seem relevant in a rule set  $R$  and have the

algorithm do the rest. The rules are specified independently from how the decisions about performing corrections are made. It is even possible to add rules that output violations, but that do not make suggestions for candidate modifications. Such rules may penalize certain constellations and can help to decide which modifications should be performed.

The following list gives some examples for rules that are useful for correcting OMR data for piano music. Each item in the list is written as a single sentence set in *italic letters* whose function is to give the reader an idea of what the rule says. Below this sentence, some comments and examples are given to further illustrate the idea of the rule. After that, a list of types of candidate modifications is given that might be proposed by the rule to eliminate or attenuate a violation. Note that the purpose of this list only is to give a rough idea about how the proposed concept works. In practice, a rule is specified as Java code that, given OMR data as an input, finds violations and creates candidate modifications that concretely specify which objects in the OMR data are to be changed what way.

(R1) *Each staff must begin with a clef.*

This rule is violated if the first (leftmost) symbol on a staff is not a clef. A violation could, for example, be caused by the OMR software misclassifying a clef as a different symbol (e.g., a note or rest) or omitting the clef completely, as depicted in the example illustration below.



Proposed data modifications:

1. Insert a clef before the first symbol. (Since we consider piano music, this can be either a treble clef or a bass clef. This yields two candidate modifications.)
2. Change the type of the first symbol on the staff to a clef (again two candidate modifications).

(R2) *Usually, the left hand staff uses a bass clef and the right hand staff uses a treble clef.*

Obviously, this rule is not very strict. Even though the situation depicted in the figure below poses a violation to this rule, the two treble clefs at the beginning of the grand staff are actually correct. In fact, this rule is to be understood as a kind of penalty that can drive the algorithm to the right decision if no stronger rule dictates which modification to follow. By using a relatively small violation cost, one can make the algorithm slightly favor a treble clef in the left hand and a bass clef in the right hand in situations where, for example, a clef is to be inserted due to a violation of (R1). By having this rule not propose any candidate modifications, one can avoid this rule to cause any unwanted changes to the data, as in the example depicted



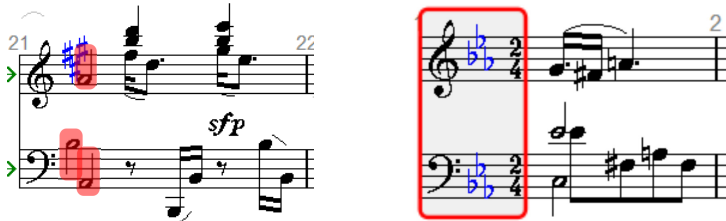
below.



Proposed data modifications: None

- (R3) *No chord/rest can precede or overlap with the horizontal region occupied by the heading clefs, key signatures, or time signatures at the beginning of a grand staff.*

In the left example depicted below, three notes violate this rule, because they overlap with a heading key signature of the upper staff. In fact, these notes are the result of the OMR software misclassifying accidentals of type “sharp” as note heads. A box in the right illustration indicates the horizontal region occupied by the heading clefs, key signatures, and time signatures, inside which no chords or rests are allowed by this rule.



Proposed data modifications:

1. Remove the violating chord/rest.

- (R4) *Certain pairs of symbols must not overlap.*

One of the most important tasks in typesetting of Western music notation is to optimize the readability for a human reader. Since overlapping symbols can pose a significant visual obstruction, there are best practices for minimum spacings between certain types of symbols. Even though slightly touching symbols can appear in some cases, situations as depicted below would pose a clear violation of this rule.

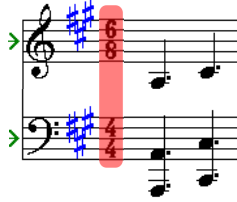


Proposed data modifications:

1. Remove/move one of the involved symbols.

- (R5) *Time signatures in left hand and right hand staff must match.*

In the example below, the left hand staff declares a 6/8 time signature, while the right hand staff declares a 4/4 time signature, which is considered a violation of this rule.



Proposed data modifications:

1. Change the left hand time signature.
2. Change the right hand time signature.

(R6) *Voice durations must match the time signature.*

In most cases, the durations of the chords and rests within each voice contained in a bar are expected to add up to the duration declared by the time signature that is valid for that bar. Exceptions to this rule are offbeat bars that can occur at the beginning and end of a musical section and voices that begin or end in the middle of a bar (see Figure 5.3). In the example depicted below, a missing beam causes the duration of the voice in first bar of the lower staff to add up to 6/4 instead of just 2/4 as suggested by the time signature.



Proposed data modifications:

1. Modify the duration of a chord/rest in the voice.
2. Insert or remove a chord/rest.
3. Modify voices/membership in voices.
4. Modify the time signature.
5. Insert/remove a time signature.

(R7) *Membership in the same raster group implies having the same onset time.*

If the onset time of chords is determined by the duration of the preceding chords and rests in the same voice, errors in the detected durations or missed out chords or rests may cause this onset time to drift apart for different voices in the same raster group. An example is depicted below. Here, at the beginning of Voice 2, a beamed group of 4 eighth notes has erroneously been recognized as 3 individual quarter notes. This causes violations for this

rule at three raster groups which are highlighted in the picture. In the first violation, the onset time of the note in Voice 1 is  $1/4$  but the onset time in Voice 2 is already  $2/4$ . The error propagates to all subsequent raster groups where other voices share a member with Voice 2.

Proposed data modifications:

1. Modify the duration of a chord/rest that precedes the violation to adjust the onset time.
2. Modify voices/membership in voices.
3. Modify raster groups/membership in raster groups.

(R8) *No two chords/rests of a single raster group can be member of the same voice.*

The voice assignment depicted in the example below would cause two violations for this rule, because Voice 1 has two member chords/rests at the highlighted raster groups.

Proposed data modifications:

1. Modify voices/membership in voices.

(R9) *Each chord/rest in a raster group should belong to one and only one voice.*

A voice assignment as illustrated in the example below would cause three violations to this rule. Two highlighted chords are not assigned to any voice.

One chord is assigned to two voices simultaneously.

Proposed data modifications:

1. Modify voices/membership in voices.

(R10) *Key signatures in left hand and right hand staff must match.*

In the example below, three of four  $b$ s of the key signature in the right hand have not been recognized. This leads to a violation, because now the left hand staff declares a  $4b$  key signature, while the right hand staff declares a  $1b$  key signature.

Proposed data modifications:

1. Change the left hand key signature.
2. Change the right hand key signature.

(R11) *If a key signature changes at the beginning of a new line, this change must be “announced” at the end of the previous line.*

An example of such an “announced” key signature change is depicted in Figure 5.9. A case, where this rule is violated is depicted below. This violation is caused by the OMR software missing out the same amount of accidentals from the key signatures at the beginning of the second line.

Proposed data modifications:

**Figure 5.9.** Example for a key signature change that is “announced” at the end of a grand staff.

1. Alter the key signature at the beginning of the new line, so that there is no longer a change in key.
2. Alter the key signature at the beginning of the previous line, so that there is no longer a change in key.
3. Insert an “announcing” key signature change at the end of the previous line.

(R12) *There should be no unnecessary empty space that stretches across all staves of a grand staff.*

Empty space, as highlighted in the example below, is usually avoided in sheet music typesetting. In the example, the space is caused by a key signature change showing 4 naturals that has been missed out by the OMR software.

Proposed data modifications:

1. Insert a symbol inside the empty space. (Since there are too many choices for symbols that could be added inside empty space, this rule might best be combined with suggested modifications of other rules that propose the insertion of a particular symbol.)

Rules (R1)-(R4) can fix missing clefs and remove objects that lie in the scope of the staff header and have erroneously been classified as chords/rests such as is the case in the example shown in Figure 4.3. Using rules (R5)-(R9) one might be able to simultaneously find the most likely voice assignments, fix broken time signatures, and correct errors in chord/rest durations. Since all of these aspects are interdependent, the combined optimization of all these aspects

can yield better results than optimizing each aspect individually. Using rules (R10)-(R12) the key signatures can be fixed even beyond the level of what the module described in Subsection 5.2.2 does. Using the combination of rules, it becomes possible to fix cases where the key signature of both the left hand and right hand staff have been recognized incorrectly. It even becomes possible to infer the insertion of key signatures that have been missed out by the OMR software by following a suggested insert operation of rule (R11).

The approach is rather naively trying each candidate modification and chooses to perform the one with the biggest improvement. This may be seen as a steepest decent method for minimizing a high-dimensional inconsistency function. As is always the case with such methods, there is the chance of getting stuck in a local minimum of the function to minimize. In our case this would mean that after the algorithm has finished, a lower inconsistency could still be achieved, but it would require performing more than one modification in a single step to reach it.

The shape of the inconsistency function depends on which rules are included in the rule set and on the tuning of the violation cost produced by these rules. Some rules are known to be very strict, e.g., (R3), so a violation of that rule should produce a relatively high cost. Other rules are simply saying that certain constellations are more likely than others, but with exceptions occurring from time to time. In such cases, the cost for a violation should be set to a lower value. While adding rules for this approach is rather simple, tuning the violation cost so that the inconsistency function indeed has its global minimum at the desired configuration of the data is a delicate task. However, by carefully tuning violation cost and adding rules for expressing even minor preferences and details of music notation, some local minima can be eliminated.

The major drawback and limitation of this approach is its runtime behavior. Naively trying out each candidate modification and recalculating the violations for each try leads to an expected computational cost that scales with the cube of both the length of the input OMR data and the number of rules, see Appendix A. Obviously, the algorithm wastes a lot of resources by recalculating the violation cost for the complete data when checking the improvement of a particular modification. This might in fact not be necessary, because a modification might only affect violations in some local neighborhood. To speed up the algorithm, one should introduce some sense of locality to avoid unnecessary calculations of violation cost at locations that have no effect on the location of change. If the local neighborhood that affects violations can be assumed to be of constant size in general, this would reduce the complexity from being cubic to being quadratic in the length of the input OMR data.

Using suitable collections of rules, many OMR errors can be detected and corrected. However, there clearly is a limitation to what can be fixed. In some cases, the OMR data might be obstructed beyond repair through postprocessing. For example, missing notes are hard to infer from rules only. Even if it can be inferred that some object occupying time in a voice needs to be inserted, it will be hard to choose between inserting rests, notes, or chords with different pitches. Some rules regarding continuation of musical patterns or melody lines suited to the underlying musical style could be used to assess which options are more likely than others. However, in extreme cases, one would end up having to guess complete sections of a musical composition.

Instead of inferring corrections based on erroneous OMR results, a much more powerful approach would be to incorporate the consideration of high-level rules such as those listed above directly into the OMR process. This way, the rules could be used to increase the certainty of decisions on the recognition and classification of symbols and semantics *before* erroneous decisions are made and important information is lost. The main problem with putting this idea into practice is that rules are spread across very different semantic levels. For example, some rules might work on the level of pixels, others might work on the level of shapes and symbols, and again others might work on note events and voices. However, without having already derived shapes and symbols, one cannot apply the rules that work on this level, and without having already derived note events and voices, one cannot apply the rules that work on that level, too. Clearly, all of these levels are strongly interconnected, and to infer decisions with high certainty, all of these levels have to be taken into account simultaneously.

## 5.4 Detecting Repeats and Jumps

At this point, the symbolic score data obtained from the OMR results do not contain any information other than the plain notation bar sequence  $\nu$  about the sequence in which the bars are to be played. However, for converting the symbolic score data to note events, we want to use the sequence of bars as it is suggested by the score for a performance, i.e., we want to find out the default bar sequence  $\delta$ . Differences between the default bar sequence  $\delta$  and the notation bar sequence  $\nu$  are caused by repeats and jumps. Since repeats can be seen as a special case of jumps, we will only speak of jumps in the following. In general, due to OMR errors, we will not be able to always find the correct default bar sequence from the sheet music. Therefore, the goal pursued in this section is to derive an estimate  $\delta^*$  that is a close to  $\delta$  as possible. The general approach to calculate  $\delta^*$  is to start with the first bar of the track, i.e., the first bar in  $\nu$  and then follow  $\nu$  until some symbol or text indicates that a jump to a different bar than the successor in  $\nu$  is to be performed. We will discuss the possible kinds of jumps in the following. Note that in order to represent  $\delta^*$ , it is sufficient to store a list of jumps that is to be performed with respect to the notation bar sequence  $\nu$ . Besides these jumps,  $\delta^*$  always follows the notation bar sequence  $\nu$ . A jump is specified by a jump source  $\nu_s \in \nu$  and a jump target  $\nu_t \in \nu$  with  $t \neq s + 1$ .

The most simple type of jump is a repeat indicated by repeat signs. An example is illustrated in Figure 5.10(a). When reaching the bar with closing repeat signs, one has to jump back in  $\nu$  until either to the closest bar with opening repeat signs or the beginning of the track or a titled subsection of the track is reached, see also Figure 5.10(b). A regular repeat sign indicates a single repeat only and, therefore, is ignored the second time it is reached. The recognition of repeat signs by the OMR software is quite reliable. In a test dataset of 5 sheet music books consisting of piano sonatas by Beethoven, Mozart, and Schubert, as well as string quartets by Mozart, 564 of 570 repeat signs were recognized correctly, which is a rate of about 98.95%. The cases, where the repeat signs are not recognized by the OMR software, are usually either caused by the repeat signs being surrounded by parantheses or the dots of the repeat signs touching nearby staff lines, see Figure 5.11.

(a) Simple repeat indicated by opening and closing repeat signs. The default bar sequence  $\delta$  is shown as a single solid line covering the entire staff.

(b) Example for a titled subsection acting as a jump target for a simple repeat. The staff is divided into two sections: "Scherzo" and "Trio". The default bar sequence  $\delta$  is shown as a solid line for the Scherzo section and a dashed line for the Trio section.

(c) Simple repeat with two alternative endings indicated by number brackets. The staff is divided into two sections: "1." and "2.". The default bar sequence  $\delta$  is shown as a solid line for the first section and a dashed line for the second section.

(d) A repeat with 4 runs and 3 alternative endings. The staff is divided into four sections: "1. 2.", "3.", and "4.". The default bar sequence  $\delta$  is shown as a solid line for the first section and dashed lines for the second, third, and fourth sections.

(e) A da capo jump with a "fine" ending. The staff is divided into two sections. The first section ends with a double bar line and the word "Fine". The second section ends with a double bar line and the word "D.C.". The default bar sequence  $\delta$  is shown as a solid line for the first section and a dashed line for the second section.

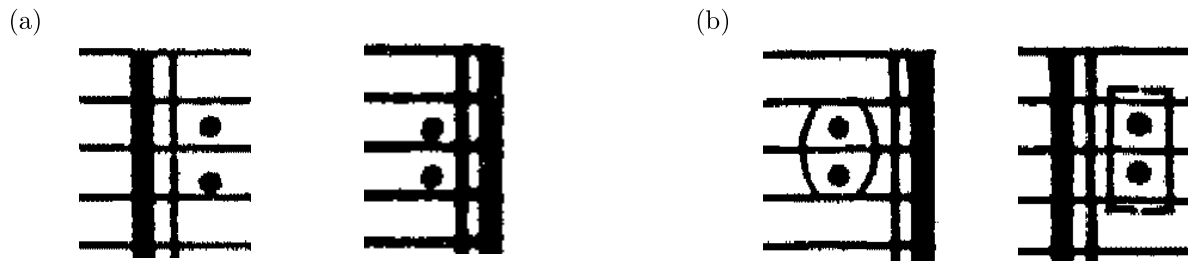
(f) A dalsegno jump with a "fine" ending. The staff is divided into two sections. The first section ends with a double bar line and the word "Fine". The second section ends with a double bar line and the word "D.S. al Fine". The default bar sequence  $\delta$  is shown as a solid line for the first section and a dashed line for the second section.

(g) A da capo jump and a coda jump. The staff is divided into two sections. The first section ends with a double bar line and the word "D.C. al Coda". The second section ends with a double bar line and the word "Coda". The default bar sequence  $\delta$  is shown as a solid line for the first section and a dashed line for the second section.

(h) Example for a textual jump directive that references the titles of subsections. The staff is divided into three sections: "Allegretto", "Scherzo", and "Coda". The default bar sequence  $\delta$  is shown as a solid line for the first section and a dashed line for the second and third sections.

**Figure 5.10.** Illustration of several types of repeats and jumps in sheet music. For each example, the resulting default bar sequence  $\delta$  is indicated below the staff by a folded arrow with solid and dashed sections. The solid sections indicate bars that are to be played and the dashed sections indicate jumps. (a) Simple repeat indicated by opening and closing repeat signs, (b) example for a titled subsection acting as a jump target for a simple repeat, (c) simple repeat with two alternative endings indicated by number brackets, (d) a repeat with 4 runs and 3 alternative endings, (e) a da capo jump with a “fine” ending, (f) a dalsegno jump with a “fine” ending, (g) a da capo jump and a coda jump, (h) example for a textual jump directive that references the titles of subsections.





**Figure 5.11.** Examples of double bar lines with repeat signs that are mistakenly recognized by the OMR software as plain double bar lines without repeat signs. In the case of type (a), the recognition error is probably caused by one of the dots being slightly misaligned and touching the staff line below. In the cases of type (b), the recognition of the repeat signs probably fails because of the additional brackets.

Our algorithm for finding jumps induced by repeat signs works as follows. Stepping through the notation bar sequence  $\nu$ , we always keep track of the most recent candidate jump target  $\nu_t$ . A bar  $\nu_t$  can serve as a candidate jump target if it is the first bar of the track, the first bar of a new titled subsection (e.g., “Trio” in a track that consists of a section titled “Scherzo” followed by a section titled “Trio”), or a bar with opening repeat signs. Once a closing repeat sign is reached at bar  $\nu_s$ , we add a jump with source bar  $\nu_s$  and target bar  $\nu_t$  to the list of jumps. Unfortunately, titled subsections are hard to recognize from the OMR results because the recognition of text that might serve as title is very unreliable and often it is not clear whether a such a text is to be interpreted as a title, a tempo directive, or even both. An example for a subsection title can be seen in Figure 5.9. Therefore, instead of relying on the recognition of titled subsections, we make the assumption that the jump target of a repeat never lies before the closing repeat sign of a previous repeat. Using this assumption, we simply add all bars that immediately follow closing repeat signs in  $\nu$  as candidate jump targets. In our tests, this method did not only cover the cases of titled subsections, but was also able to repair cases where opening repeat signs had not been recognized. In total, 350 of 353 jumps that are induced by repeat signs were recognized correctly on the test dataset, which is a rate of 99.15%.

A second type of jump is induced by repeats with alternative endings. Such alternative endings are usually indicated by number brackets above the bars of the alternative sections, see Figure 5.10(c). In most cases, there is a single repeat with two alternatives where the first alternative (indicated by the number 1) is to be played in the first run and the second alternative (indicated by the number 2) is to be played in the second run. This manifests as one additional jump besides the one induced by the repeat signs that is performed to skip the first alternative in the second run of the repeated section. When reaching the bar  $\nu_s$  preceding the first alternative in the second run, a jump has to be performed to the bar  $\nu_t$  that is the first bar of the second alternative. Despite the cases of having two alternatives and a single repeat, there can be cases with more than two alternatives and more than a single repeat, see Figure 5.10(d). In the test dataset, a total of 97 repeats with alternative endings are found. Unfortunately, the OMR software used in this project did not recognize the number brackets that indicate alternative endings at all. Therefore, this type of jump cannot be recognized and taken into account in our calculation of  $\delta^*$ . In Chapter 7, however, we briefly outline a strategy for trying to capture this type of jump through content-based

comparison.

As a third category, we consider jumps that appear in the context of so-called *da capos* and *dal segnos*. *Da capo* is Italian for “from the beginning” and instructs the player to jump back to the beginning of the track to play another run, which we call *da capo run*. Usually, in the *da capo* run, repeat signs are ignored and for repeats with alternative endings, the last alternative is chosen, see Figure 5.10(e). *Dal segno* is Italian for “from the sign/marker”. This directive appears in combination with special marker symbols such as  $\%$  or  $\Phi$  that specify the bar that acts as the jump target in the *dal segno* jump, see Figure 5.10(f). The  $\Phi$  symbol, furthermore, is often used to indicate a jump from a *da capo* run to a so-called “Coda” section that acts as the final section of the track, see 5.10(g). *Da capo* and *dal segno* jumps are often indicated by textual directives, jump marker symbols, or a combination of the two. Textual jump directives are usually written below the intended jump source bar, which is expected to end with some sort of double bar line indicating the end of the piece or a musical subsection. The directives for *da capo* and *dal segno* jumps can be written in many ways. Often, abbreviations are used such as “D.C.” for *da capos* and “D.S.” for *dal segnos*. *Da capo* and *dal segno* jump directives can be extended by additional parameters. Using the word *fine*, which is Italian for “end”, a piece can be declared to end at a bar that is different from the last entry in the notation bar sequence  $\nu$ . An example usage of this keyword can be found in Figure 5.9 and in Figure 5.10(e)-(f). In such a case, the target end bar is marked by the keyword “fine” and the textual jump directive might read, for example, “*da capo al fine*”. However, the “fine” marker can also be used to indicate the end of the piece without being explicitly mentioned in the textual jump directive. Textual jump directives can use subsection titles as parameters. For example a directive “Menuetto D.C.” indicates that the subsection titled “Menuetto” is to be repeated from the beginning. A single textual directive can, furthermore, imply more than a single jump. For example, the directive “Allegretto D.C. e poi la Coda” implies, first, a jump back to the beginning of the subsection titled “Allegretto” and, second, a jump from the end of the Scherzo to the beginning of the subsection titled “Coda”, see Figure 5.10(h). Note that some of the indicators for jumps are bound to certain conditions. For example, if a bar contains both a repeat sign and a *dal segno* directive, as in Figure 5.10(f), the *dal segno* directive is to be followed only in the second run of the repeat. As another example, the coda jump illustrated in Figure 5.10(g) is only to be followed in the *da capo* run.

To detect *da capo* and *dal segno* jumps from the OMR results, the detected text elements are searched for keywords such as “*da capo*”, “d.c.”, “*fine*”, and “*coda*”. On the test dataset, 20 of 29 *da capo* jumps and 11 of 15 cases where the track ended at a “*fine*” keyword were correctly recognized. In the remaining cases, either the OMR software failed to recognize the existence of text at all or the keyword was obstructed by errors in the recognized text. Experiments on the test data showed that using the edit distance for a fuzzy comparison to account for possible inaccuracies in the text recognition of the OMR software did not significantly improve the results without causing false positives, i.e., song lyrics or other text erroneously being classified as one of the keywords. The detected *da capo* jumps and “*fine*” endings are taken into account in our estimated default bar sequence  $\delta^*$ . Since *segno* and *coda* marker symbols are not detected by the OMR software, the corresponding jumps are not taken into consideration in  $\delta^*$ .

As a final category of jumps, we consider cases that do not fall in any of the three categories

The image shows a musical score for a piece titled "MENUETTO". It consists of four staves. The first staff is the title. The score is divided into three sections by dashed lines. The first section ends with a double bar line and a fermata above and below. The second section begins with a double bar line and a fermata above and below. The third section ends with a double bar line and a fermata above and below. The text "Attacca il Menuetto subito" is written below the final staff.

**Figure 5.12.** Example for a da capo jump and a “fine” ending similar to Figure 5.10(e) that, however, does not use the expected keywords. Instead of “da capo” or “D.C.” the text “Attacca il Menuetto subito” equivalently directs the performer to jump back to the beginning of the section titled “Menuetto”. Instead of using the keyword “fine”, the end of the da capo run is marked by fermata symbols above and below the target double bar line.

discussed above. This category covers exceptional cases and free-form textual directives that are very hard to interpret computationally. A nice example of such a case is depicted in Figure 4.14, where a jump marker is used together with a textual description in four different languages that points out under what condition a jump to a matching marker found elsewhere in the score is to be performed. A somewhat less extreme example that, however, still falls outside the expected conventions is the use of the textual directive “Attacca il Menuetto subito” to indicate a da capo jump without using any of the corresponding keywords, as seen in Figure 5.12. Furthermore, fermata symbols above and below a double bar line are used instead of the “fine” keyword to mark the end of the track after the da capo run. As a last example, consider the situation depicted in Figure 5.13, where a pair of  $\%$  markers is used to indicate that a section should be repeated multiple times (one time for each verse of the lyrics). Furthermore, a fermata in the last bar is used to imply the piece is supposed to end at that position within the bar after the last verse is played. Cases of this category are not taken into consideration in our automatic computation of  $\delta^*$ .

## 5.5 Estimation of Onset Times and Tempo

In this section, we focus on how to extract temporal information such as onset times and durations of note events from the OMR results. Since in common Western music notation, the score is usually divided into bars that are played consecutively, it is sufficient to determine the onset times and durations relative to the beginning of each bar individually. The absolute onset times can then be derived by concatenating the bars according to a suitable bar sequence, such as  $\delta^*$  derived in the previous section. Onset times and durations are usually measured in musical units such as full notes, half notes, and quarter notes. The musical duration of notes and rests is specified by their shape and by the attachment of beams or flags to note stems. How these musical units translate to physical units such as seconds is

The figure shows a musical score for a song. The top part is the Singstimme (Vocal) and the bottom part is the Pianoforte (Piano). The tempo is marked 'Mäßig geschwind' and the piano part is marked 'mf'. The lyrics are: 'Wan - dern ist des 2. Was - ser ha - ben 3. sehn wir auch den 4. Stei - ne selbst, so 5. Wan - dern, Wan - dern, 2. Vom 3. Das 4. Die 5. 0'. A dashed line connects the first verse to the subsequent verses. A dotted line indicates the end of the track. A fermata sign is placed at the end of the last verse in both parts.

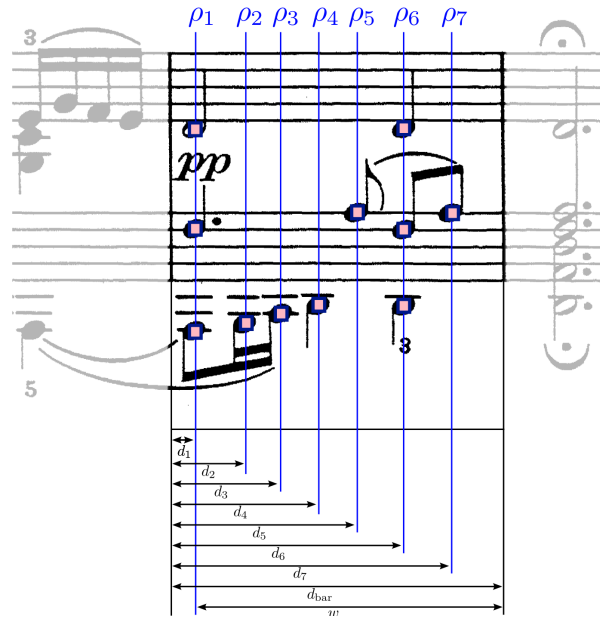
**Figure 5.13.** Example for a pair of markers being used to indicate multiple repeats of a verse section with different lyrics for each verse. A fermata sign in the last bar implies that, once the last verse is reached, the track is to end at this position.

determined by what is called the *tempo*. The tempo is often specified as a numerical value called *beats per minute* (BPM). A BPM value says how many subsequent quarter notes would fit in one minute of time. Given the tempo  $T$  in BPM, the duration  $s_{\text{quarter}}$  of a quarter note in seconds can, therefore, be calculated as

$$s_{\text{quarter}} = \frac{60}{T}.$$

Within a bar of music, the onset time in musical units of a note is usually determined by the duration of the preceding notes in the same voice. Unfortunately, this approach is not very robust in our scenario due to two reasons. Firstly, the recognition of beams and flags and, therefore, the recognition of note durations is not very reliable. Secondly, notes that have completely been missed out fail to contribute any duration at all. Therefore, simply adding up note durations for each voice individually to determine onset times could lead to voices drifting apart temporally. However, the aspect of simultaneity is a very important characteristic of the music. We, therefore, propose a method that tolerates some inaccuracies in relative onset times in order to robustly preserve the simultaneity of note onsets. Instead of accumulating voice durations, onset times of notes in a bar are determined by the relative horizontal position of their respective raster group and are measured not in musical units but as fractions of the bar width. Assuming that the duration of the bar is known, these fractional units can easily be converted to musical or physical units. Figure 5.14 illustrates how the fractional units are determined for an example bar of piano music. Let us assume that, in general, we have  $R$  raster groups ( $\rho_1, \dots, \rho_R$ ) which are sorted by horizontal position and with their horizontal distances from the left bar line being denoted by  $d_1 < \dots < d_R$ . Furthermore, we denote the horizontal distance of the right bar line from the left bar line as  $d_{\text{bar}}$ . Even though the leftmost raster group  $\rho_1$  is usually spaced apart from the left bar line for layout reasons, the first raster group is understood to mark the temporal beginning of the bar. Since the spacing between the leftmost raster group and the left bar line does not correspond to any temporal duration, we define the width  $w$  of the bar as the horizontal distance between the leftmost raster group and the right bar line:

$$w := d_{\text{bar}} - d_1.$$



**Figure 5.14.** Illustration of the geometric distances used for calculating onset times in fractional units.

Then, each raster group in the bar is assigned the fractional onset time

$$f_r = \frac{d_r - d_1}{w}, \quad r \in [1 : R].$$

If the duration of the bar in seconds  $s_{\text{bar}}$  is known, the onset times  $f_r$  in fractional units are simply converted to onset times  $s_r$  in seconds through

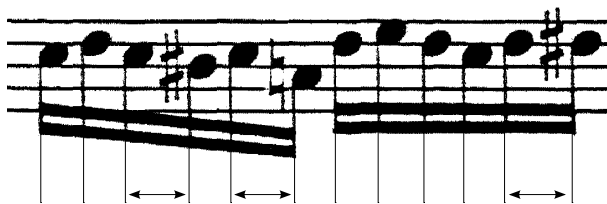
$$t_r = f_r \cdot s_{\text{bar}}.$$

In most cases, the horizontal positions and distances are roughly proportional to the musical onsets and durations within a bar. However, requirements of layout, especially rule (R4) but also rule (R12), are given priority over preserving the proportionality. This leads to inaccuracies in relative onset times. For example, in a run of short notes that are written with little horizontal distance from each other, if one of the note has an accidental, some extra space is added before that note in the layout to keep the accidental from overlapping with the previous note. This situation is depicted in Figure 5.15.

As stated earlier, the fractional units can be converted to musical or physical units when the duration of the bar in musical or physical units is known. The duration of a bar in musical units is determined by the time signature. For example, a time signature 4/4 implies a bar duration of four quarter notes. We denote the musical duration of a bar in quarter notes by the symbol  $q_{\text{bar}}$ . For a time signature with numerator  $n$  and denominator  $k$  it is calculated as

$$q_{\text{bar}} = \frac{n}{k} \cdot 4.$$

The bar duration can also be determined by the duration of the voices it contains, and, in theory, this value should agree with the value obtained from the time signature. In practice

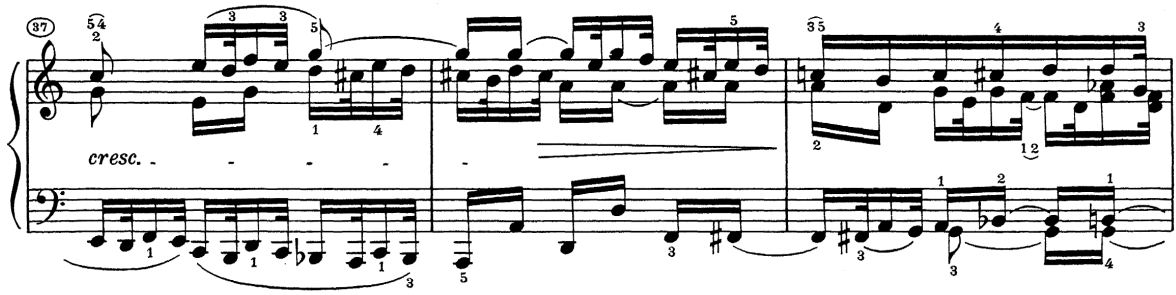


**Figure 5.15.** Notes that are usually written with little horizontal distance are spaced apart from the previous note more widely to avoid accidentals from overlapping with neighboring notes.

of our scenario, however, the recognition of voices and their durations is not very reliable, and voices in bars with multiple voices can disagree about the correct duration. We, therefore, choose to use the time signature as a criterion. The voice durations may have been already taken into account for validating or correcting the time signatures through OMR postprocessing.

To convert musical onset times and durations to physical onset times and durations, we need to determine the tempo in BPM. In practice, the scores in the repertoire of the project most often do not specify a tempo in BPM directly. Instead they use text labels that, by convention, are mapped to rough tempo classes. The vocabulary used for tempo labels depends on the individual piece and edition. Many scores use a mostly standardized vocabulary of Italian labels such as *Allegro*, *Andante*, or *Adagio*. However, in general, any language could be used and the use of conventions is optional. For example, one may encounter German tempo labels such *Geschwind*, *Mäßig*, or *Etwas Langsam*, which, without additional knowledge about context, composer, and period, will give only a very rough idea of what tempo might be appropriate. Considering this uncertainty in the vocabulary and the possible coarseness of its meaning, it is no surprise that the OMR software does not output any information about tempo at all. Textual tempo directives, if recognized at all, are output as general purpose text elements without further classification. However, for our scenario, it would be beneficial to have at least a rough estimate of the tempo, because large discrepancies between the estimated tempo of the score data and the actual tempo of the audio recording lead to additional difficulties in the content-based comparison. Unfortunately, the recognition of text elements in the OMR software is too unreliable to expect to properly solve this issue by classifying the recognized text elements to find out the tempo directives ourselves. Furthermore, due to the freedom of vocabulary and the context-dependency of its meaning, the translation from textual tempo directives to tempo estimates in BPM is a non-trivial task, which, in the context of this thesis, is seen as future work. For our scenario, we simply use a fixed tempo estimation for all of the OMR data and move the problem of dealing with the tempo differences to the content-based comparison techniques.

Experiments with the Beethoven piano sonatas, for which the recorded performances use tempos ranging from about 25 to 300 BPM, have shown that even when assuming that the performance sequence  $\pi$  is known the global synchronization using DTW can break down if the differences in tempo become too large. This behavior was observed for a passage part of which is depicted in Figure 5.16. The tempo of the recorded interpretation by Alfred Brendel uses a tempo of approximately 25 BPM for this section. Using a fixed tempo estimation for



**Figure 5.16.** A grand staff taken from the G. Henle edition of Beethoven’s Piano Sonata No. 32, Opus 111, Arietta, for illustration of the adaptive tempo estimation strategy. The time signature for this passage is 6/16. Each of the three bars has 12 raster groups. Using the adaptive tempo estimation strategy with a duration setting of 200 milliseconds per raster group, the resulting tempo is 37.5 BPM.

the OMR data of 100 BPM, the difference in tempo is a factor of 4. To deal with this issue, we introduce an adaptive tempo estimation strategy to replace the fixed tempo estimation. In the adaptive estimation, the tempo is estimated individually for each bar. The estimated tempo depends on how many raster groups are found inside a bar. Bars with many raster groups usually indicate the presence of notes with a short duration in musical units, e.g., sixteenths or shorter. Manual inspection revealed that the performances that use the extremely low tempos, like in the example case, correspond to pieces that are written using these rather short note duration. Therefore, the adaptive tempo estimation is designed to deliver a lower tempo for bars that contain many short notes in contrast to bars that contain only few notes with longer duration.

The adaptive estimation of the tempo for a bar of sheet music is performed in three steps. In a first step, the duration of the bar  $s_{\text{bar}}$  in seconds is calculated as a base duration  $s_{\text{base}}$  in seconds multiplied by the amount of raster groups  $R$ , i.e.  $s_{\text{bar}} = R \cdot s_{\text{base}}$ . In our scenario, we use a base duration  $s_{\text{base}} = 0.2$  seconds. Then, in a second step, the bar duration  $s_{\text{bar}}$  is converted to a tempo  $T$  in BPM by means of the musical bar duration in quarter notes  $q_{\text{bar}}$  specified by the time signature

$$T = 60 \cdot \frac{q_{\text{bar}}}{s_{\text{bar}}} = 60 \cdot \frac{q_{\text{bar}}}{R \cdot s_{\text{base}}}.$$

In a third step, the tempo is limited to a maximum BPM value, in our case 100 BPM, to avoid the tempo being estimated too high for bars that only contain very few raster groups, e.g., bars with only a single full note for each voice. In the example case depicted in Figure 5.16, the time signature is written as 6/16 and there are 12 raster groups per bar. This leads to a bar duration in seconds of  $s_{\text{bar}} = R \cdot s_{\text{base}} = 12 \cdot 0.2 = 2.4$ . Taking the time signature into account, the duration of a single quarter notes amounts to 1.6 seconds which is equivalent to a tempo estimation of 37.5 BPM.

If the performance sequence  $\pi = (\pi_1, \dots, \pi_K)$  is known in advance, a mean bar duration  $\bar{s}_{\text{bar}}$  in seconds can be estimated from the duration of a corresponding audio recording  $s_{\text{audio}}$  in seconds and the total number of bars  $K$  by means of the formula

$$\bar{s}_{\text{bar}} = \frac{s_{\text{audio}}}{K}.$$

When additionally knowing the time signature for each bar, the duration of the score track in quarter notes  $q_{\text{score}}$  can be derived as  $q_{\text{score}} = \sum_{k=1}^K q_{\text{bar}}(\pi_k)$  and used to compute a mean tempo  $\bar{T}$  of the score track by means of

$$\bar{T} = 60 \cdot \frac{q_{\text{score}}}{s_{\text{audio}}}.$$

Since  $\pi$  is usually not known,  $\delta$  or  $\delta^*$  can be used instead. Even though this may lead to inaccurate values for  $q_{\text{score}}$ , the error can usually be expected to be small and is very likely to be smaller than a factor of 2, which for our application is considered not critical. We will make use this strategy for the tempo estimation in Chapter 7. What remains problematic are score tracks for which the tempo varies by large factors within the track. This is, for example, the case in Beethoven’s Piano Sonata No. 26, Opus 81a, “Les adieux”, where the tempo starts with *Adagio* and then changes to *Allegro*. Clearly, for such cases, the estimation of a single average tempo for a single score track cannot be accurate.

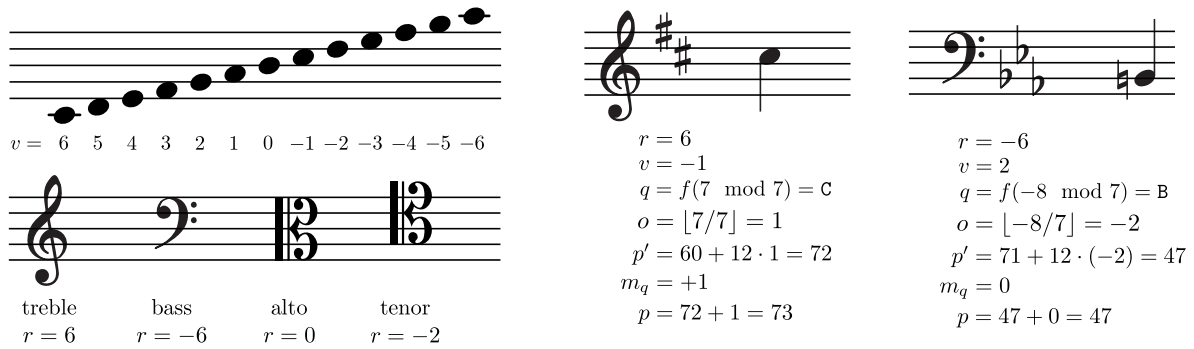
## 5.6 Deriving Pitches

Our method for determining pitches of note events derived from the symbolic score data is to follow the basic rules of Western music notation. These basic rules state that the pitch of a note event is determined by its vertical position on the staff, the currently valid clef and key signature, as well as accidentals occurring in the bar where the note event starts. For piano music, these rules are usually sufficient to derive accurate pitch information. For other types of music, in particular orchestral music, additional difficulties arise, some of which are discussed in Section 5.7. Furthermore, music notation can comprise many exceptions and conventions that make determining accurate pitch information very hard, see [15]. Throughout this section, we assume that for the type of symbolic score data given, the basic rules stated above are sufficient.

Our goal is to determine the pitches of note event within each bar in form of MIDI pitch numbers  $p \in [0 : 127]$ . A list of MIDI pitch numbers and their corresponding frequencies and musical names can be found in [147]. To determine the pitches of note events within each bar, we step through the bars of the score in order of the given bar sequence  $\delta^*$ . For each staff, we scan the corresponding symbols in left-to-right order and keep track of the symbols that affect the pitch, i.e., clefs, key signatures, and accidentals. Clefs stay valid until the end of the track unless they are replaced by another clef in the same staff before the end of the track is reached. Key signatures stay valid until the end of the line is reached unless they are replaced by another key signature before that. Accidentals directly attached to note heads are usually valid until the end of the bar. When a note symbol is found, its pitch is determined using the clef, key signature, and accidentals currently valid for this staff. To this end, the following steps are performed.

1. The vertical position  $v \in \mathbb{Z}$  of the note head is obtained from the OMR data. Here, the value  $v = 0$  means that the note head is on the middle line of the 5-line staff, see Figure 5.17. The value of  $v$  increases with lower vertical positions, e.g.,  $v = 1$  means that the





**Figure 5.17.** Top Left: Illustration of note head positions and their corresponding vertical position  $v \in \mathbb{Z}$ . Bottom Left: Examples of clefs and their corresponding reference pitch class offset  $r$ . Right: Two examples for determining the MIDI pitch  $p$  of a note given its vertical position, a clef, a key signature and possibly an accidental.

note head is between the third and the fourth line from the top,  $v = 2$  means that it is on the fourth line and so on. If the note head is above the middle line,  $v$  becomes negative.

- Using the currently valid clef, a pitch class  $q \in Q = \{C, D, E, F, G, A, B\}$  and an octave number  $o \in \mathbb{Z}$  is determined based on the vertical position  $v$ . To this end, for each clef, a characterizing reference pitch class offset  $r \in \mathbb{Z}$  must be known that determines how a vertical note position  $v$  translates into pitch class and octave. Then, the pitch class and octave number are computed as:

$$q = f((r - v) \bmod 7) \tag{5.1}$$

$$o = \lfloor \frac{(r - v)}{7} \rfloor, \tag{5.2}$$

with  $f$  being the canonical bijective mapping from the numbers  $[0 : 6]$  to the pitch classes  $f : 0 \mapsto C, 1 \mapsto D, 2 \mapsto E, 3 \mapsto F, 4 \mapsto G, 5 \mapsto A, 6 \mapsto B$ . A list of common clefs along with their reference pitch class offsets  $r$  is depicted in Figure 5.17.

- From the pitch class  $q$  and the octave number  $o$  a base MIDI pitch number  $p'$  is derived through

$$p' = g(q) + 12 \cdot o \tag{5.3}$$

with  $g$  being the following mapping from the pitch classes to corresponding base MIDI pitch numbers  $g : C \mapsto 60, D \mapsto 62, E \mapsto 64, F \mapsto 65, G \mapsto 67, A \mapsto 69, B \mapsto 71$ .

- Each pitch class  $q \in Q$  can be modified by both the accidentals of the key signature and individual accidentals that are attached to note heads. Each accidental affects one particular pitch class. Which pitch class that is can be determined by their vertical position in the same way as for note heads. If both the key signature and an individual accidental affect the same pitch class, the individual accidental overrules the accidental of the key signature. To take the key signatures and accidentals into account in our calculation of pitches for note events, we keep track of the currently valid pitch modifier

$m_q \in [-2 : 2]$  for each pitch class  $q \in Q$  while stepping through the symbols in a strictly left-to-right order. The list below shows the commonly used accidental symbols and their corresponding pitch modifiers  $m_q$ :

$\sharp$ (sharp)	$\mapsto$	+1
$\flat$ (flat)	$\mapsto$	-1
$\natural$ (natural)	$\mapsto$	0
$\sharp\sharp$ (double sharp)	$\mapsto$	+2
$\flat\flat$ (double flat)	$\mapsto$	-2.

To determine the MIDI pitch  $p$  of a note event, the pitch modifier  $m_q$  is added to the base MIDI pitch number  $p'$

$$p = p' + m_q. \tag{5.4}$$

Two examples for determining the MIDI pitch  $p$  of notes based on their vertical position, the clef, the key signature and possibly accidentals are illustrated in Figure 5.17.

## 5.7 Orchestral Scores

Compared to scores for piano music, OMR results of orchestral scores raise additional difficulties for extracting accurate pitches and onset times for generating note events. An example page of orchestral music is depicted in Figure 5.18. One of the difficulties in orchestral music is the possible presence of transposing instruments. These are instruments for which the nominal pitch that is written in the score differs from the pitch that sounds when the instrument is played. Common examples for transposing instruments are clarinets, horns, and trumpets. Usually, a transposing instrument is declared to be written in a certain key. The declared key specifies the pitch that sounds when the pitch  $C$  is written. From this information, one can determine the amount of semi-tones by which the written pitch must be shifted to match the sounding pitch. For example, for a clarinet written in  $B\flat$ , the written pitch  $C$  sounds as a  $B\flat$ , which is two semitones below the  $C$ . This means that, as long as the transposition does not change, all pitches for that clarinet sound two semitones lower than they are written. Throughout this thesis, we specify transpositions by giving the amount of semitones that need to be added to the written pitch in order to get the sounding pitch. For example, the clarinet written in  $B\flat$  would have a transposition of  $-2$ .

The declaration of the key for transposed instruments specifies the transposition only up to the ambiguity of octave shifts. For example, instead of a transposition of  $-2$ , an instrument written in  $B\flat$  could as well have a transposition of  $+10$ , or  $-14$ , or  $-2 + 12k$ , for  $k \in \mathbb{Z}$ , in general. The particular octave shift  $k$  depends on the particular instrument class and is assumed to be known by the reader of the score. To our application of comparing note events to audio recordings, the octave shift is not critical, because the chroma features used for the comparison discard this information anyway. However, the non-octave part of the

transposition is critical to our application, because using systematically wrong pitch classes will systematically compromise the similarity between the chroma features created from the note events and the chroma features extracted from the audio recordings.

In addition to the issue raised by transposing instruments, orchestral scores often suffer more strongly from recognition errors of symbols such as key signatures and clefs than piano music. Reasons for that might be that printed staff sizes of orchestral scores tend to be smaller, which causes the degrading effect of printing resolution and artifacts to be stronger. Another reason could be different contrast and thresholding techniques in the conversion to black and white. Certainly, these effects depend on the printing quality of the particular score. A side-by-side comparison of image quality for an extract of piano music with higher quality and an extract of orchestral music with lower quality is depicted in Figure 5.19. Note that the issues caused by printing quality cannot be solved by simply increasing the scanning resolution. Another problem of OMR that becomes more relevant with orchestral scores is the grouping of staves to grand staves. While in piano music, each grand staff always consists of two staves, in orchestral scores, grand staves are usually far more complex. Depending on the kind of orchestration, the amount of staves in a grand staff can easily exceed 20 or even pass 30. Furthermore, the selection of instruments that are included in a grand staff can change with each new grand staff. Often, the staves for instruments that do not play any notes throughout a grand staff are simply omitted to save space and to avoid unnecessarily printing staves that contain nothing but rests. The elevated complexity of grand staves in orchestral scores leads to OMR errors in the grouping of staves to grand staves becoming a relevant issue for our application. Errors in the grouping of staves usually manifest as grand staves erroneously being split horizontally into two or more smaller grand staves, see also Figure 5.18. The effect on the resulting OMR data is that in place of the subsequence of the notation bar sequence  $\nu$  that would resemble the original grand staff, we obtain multiple repeats of this subsequence with each repeat being played by a different subset of instruments. Not only does this lead to compromised note events for this subsequence, but it would also be very confusing for users to look at in the application of highlighting recognized bars in the sheet music.

To encounter the challenges posed by orchestral scores through postprocessing, we need to identify rules that can help us to detect and to fix errors based on the OMR results. In fact, there are some rules which make this task seem possible. Here are some examples:

(OR1) *Certain combinations of instrument groups and transpositions are more common than others.*

For example, french horns always use a transposition of  $-7$  (F), clarinets usually use  $-2$  (B $\flat$ ),  $-3$  (A) or  $+3$  (E $\flat$ ), and string instruments such as violins and celli use a transposition of  $0$  (C). More information on which instruments commonly use which transpositions can be found in the literature such as [69].

(OR2) *The transposition key is usually indicated as part of the staff labels that are attached to the first grand staff of each score track.*

Some examples found on the page depicted in Figure 5.18 are “2 Klarinetten in B”, “Baßklarinette in A”, “4 Hörner in F”, and “2 Trompeten in B”.

**I**  
**Inferno.** Franz Liszt.

**Lento.**

**Kleine Flöte.**  
**2 Große Flöten.**  
**2 Hoboen.**  
**Englisches Horn.**  
**2 Klarinetten in B.**  
**Baßklarinetten in A.**  
**2 Fagotte.**  
**4 Hörner in F.**  
**2 Trompeten in B.**  
**2 Tenorposaunen.**  
**Baßposaune u. Tuba.**  
**Pauken in D. A.**  
**Pauken in F. C.**  
**Becken.**  
**Große Trommel**  
*mit Paukenschlägeln (with drumsticks)*  
*(avec baguettes de timbales (üstdobverövel))*  
**Tamtam.**  
**Harfe.**  
**1. Violinen.**  
**2. Violinen.**  
**Bratschen.**  
**Violoncelle.**  
**Kontrabässe.**

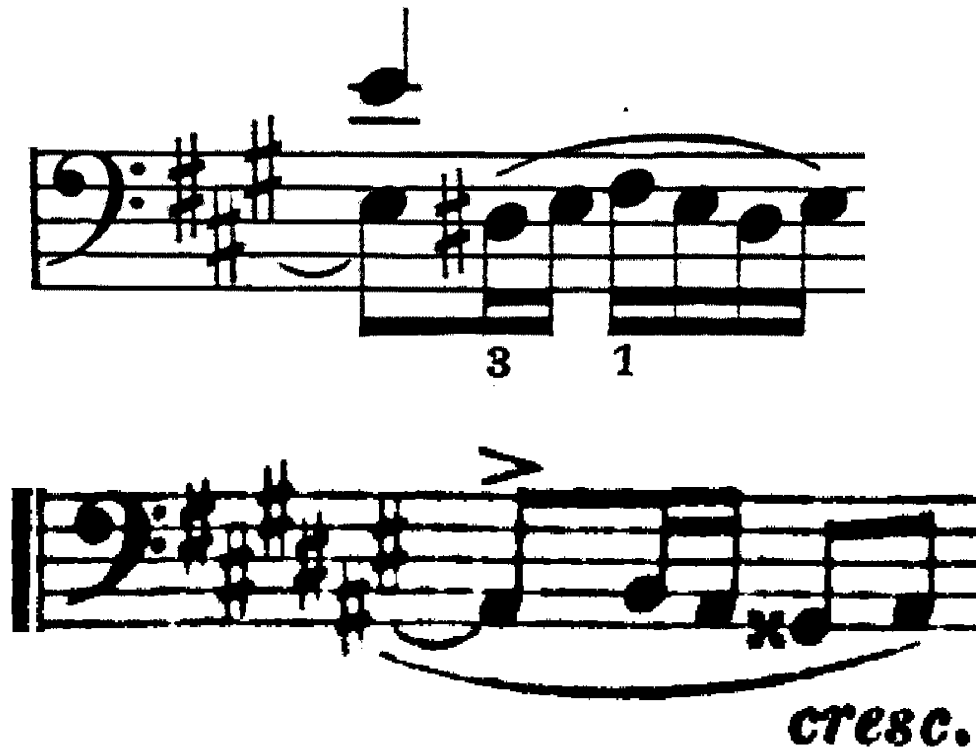
*Per me si va nella cit.ta do.len.te: Per me si va nell'e.ter.no do.lo.re:*

*gleich dämpfen (mute immediately)*  
*(sec) (hirkelen el/objant)*

**Lento.**

Mit und Druck von Breitkopf & Haertel in Leipzig **F. L. 13.**

Figure 5.18. Example page of an orchestral score showing the beginning of the Breitkopf & Haertel edition of Liszt’s Dante Symphony. This score uses special staves with only one staff line for the percussion instruments. A special problem for the OMR software is posed by the text interrupting the vertical line that marks the left boundary of the grand staff. In the OMR recognition results, this grand staff is erroneously split into two consecutive grand staves with the first one containing all staves from the “Kleine Flöte” to the “Pauken in F. C.” and the second one containing all staves from the “Harfe” to the “Kontrabässe”. The one-line staves for the percussive instruments are not recognized by the OMR software.



**Figure 5.19.** Side-by-side comparison of image quality for an extract of piano music with higher quality (top) and an extract of orchestral music with lower quality (bottom).

(OR3) *The transposition key of an instrument group is usually constant throughout a score track.*

Even though this might hold in the majority of cases, exceptions to this rule are quite common. An example of changing transpositions is depicted in Figure 5.20.

(OR4) *The first grand staff of each score track includes staff labels declaring which staves are to be played by which instrument groups.*

An example for this is shown in in Figure 5.18.

(OR5) *Even though staves for certain instrument groups can be omitted, the initial order of the instrument groups does never change throughout a score track.*

This is a very powerful rule, because, if the correct instrument groups are known for only some of the staves in a grand staff, it allows us to infer or at least narrow down the instrument groups of the remaining staves.

(OR6) *Differences in transposition can manifest in differences in key signatures between instrument groups.*

For example, having a key signature with 2 sharps in staff *A* and a key signature with 3 sharps in staff *B* might indicate, that the transposition used in staff *B* is a perfect fifth lower ( $-7$ ) than the one in staff *A* (see the top two staves in Figure 5.21). However, this rule has to be used with caution, because for some instruments, key signatures might be omitted completely in favor of using accidentals directly with each note (as in the sixth line in Figure 5.21).

(OR7) *Certain instrument groups only use a particular clefs.*

For example, violins always use a treble clef and piano only uses treble or bass clefs.

(OR8) *Each instrument group has a known limited pitch range.*

For some instruments the pitch range is limited in both directions, e.g., on a standard 88-key piano, the lowest pitch is  $A''$  (also:  $A_0$ , MIDI number 21) and the highest pitch is  $c''''''$  ( $C_8$ , MIDI number 108). For many other instruments, only a lower pitch boundary can be properly specified. For example, the lowest pitch of a violin is  $g'$  (also:  $G_3$ , MIDI number 55).

(OR9) *If the selection of instrument groups in a grand staff is different from the previous grand staff, the new selection is indicated by staff labels attached to the new grand staff.*

An example for this can be seen in Figure 5.21. Here, the staff labels are actually abbreviations of the instrument names.

The above rules show that transposition, key signatures, and clefs are all connected through the mapping between staves and instrument groups. Even though these rules suggest the possibility of finding this mapping automatically in postprocessing of the OMR data, the chance of succeeding in this strongly depends on the completeness and reliability of the OMR output. In particular, the application of the above rules relies heavily on the accurate recognition of staff labels. Unfortunately, the OMR software used in this project is very unreliable regarding the recognition of staff labels. Labels that are printed on the left side of staves are usually not recognized at all, and abbreviated staff labels that are often found above or between staves are often missed out or very low in accuracy of the recognized text. Since the recognition of key signatures and clefs is also not very reliable, either, rules (OR6) and (OR7) cannot help much to improve the situation. Furthermore, there can be exceptions to many of these rules. For example, regarding (OR2), for certain instrument groups no transposition key is written and a standard transposition for that instrument group is assumed. Regarding (OR3), there are cases where the transposition does change for certain instrument groups within a track, e.g., in the Breitkopf & Haertel edition of Liszt's Faust Symphony, see Figure 5.20. The relationship between transposition and differences in key signature is not very strict. Instead of modifying key signatures according to the transposition, composers or editors can choose to completely omit key signatures for some instrument groups and use individual accidentals for each note instead. Furthermore, certain instruments, e.g., timpani have special conventions regarding pitch and transposition. Exception to rule (OR9) can also occur in practice. In such cases, the selection and order of instruments groups included in a grand staff is assumed to be implicitly understood by means of the other rules and through similarity to previous types of grand staves.

**Figure 5.20.** Example for changes in transposition that take place within a score track. The changes are indicated by textual instruction starting with “mute in”, which means “change to”. In the example, three changes are indicated. In the third staff from the top, the clarinets that originally were written in C change their transposition key to A. In the 5-th and 6-th staff, the horns change from F to E. Such changes are usually achieved by changing either a crook or the complete instrument.

The above considerations lead to the conclusion that the recognition of transposing instruments and the correction of key signatures, clefs, and grand staves, in our scenario, cannot be achieved reliably through postprocessing only. A successful approach to this task will require incorporating more advanced OMR software or combining current OMR results with further analyses of the original image data. This leads to interesting research topics, which we get back to in our discussion of future work in Chapter 9. Instead of trying to solve all of these issues fully automatically, we pursue a pragmatic method for handling the issues by a computer-assisted manual correction of the most critical errors. Using this method, we enable the integration of orchestral scores into the workflow of the project, and we lay a foundation for automatic approaches to replace the manual steps in the future. The experience and ground truth data that is collected with our pragmatic approach will prove useful in the development of automatic approaches. In the above discussion of properties of orchestral scores, we have identified that important characteristics of the staves on each page are the grouping to grand staves, transposition, as well as the clefs and key signatures found at the beginning of each staff. For each page, we denote these characteristics as the *staff signature*. An example for staff signature information can be found in Figure 5.21.

In our pragmatic approach, staff signature information is extracted automatically and saved to a plain text file for each page of the score. Each staff found on the page is represented by a single line of information in the text file. Grand staves are separated by an empty line. Each line that represents a staff comprises columns for transposition, clef, and key signature. Optionally, additional information can be given, e.g., a flag indicating that two staves act as a single coordinate system such as for piano or harp, an identifier for the instrument group to which the staff belongs, or information about brackets grouping certain instrument groups at the left border of grand staves in orchestral music. The automatically generated files can be corrected manually by comparison with the original sheet music image. After that, the corrected staff signatures can be reimported to the OMR data, and any changes made, such as merging erroneously split grand staves or adding missing transposition information are applied automatically. Unfortunately, the manual correction of the staff signatures still take a lot of effort. The correction of score books of about 150 pages took several hours of manual

Clef	Key Signature	Transposition
treble	+2	0
treble	+3	-7
treble	-1	-3
treble	+4	-2
tenor	+2	0
treble	0	-7
tenor	+2	0
alto	+2	0
bass	+2	0
bass	+2	0
bass	+2	0

**Figure 5.21.** Staff signature annotation for an example grand staff taken from a score of the “Symphony to Dante’s Divina Commedia S109 - Inferno” by Franz Liszt. Positive key signature values count the number of sharps, negative values count the number of flats. Transposition values are specified as the amount of semitones the pitch has to be modified with to sound correctly.

work. With the use of manually corrected staff signatures, the accuracy of the pitches of the note events extracted from the OMR data is expected to be sufficient for our scenario. The remaining inaccuracy will mostly be caused by individual accidentals being missed out, which is not expected to affect enough note events to be considered critical.

Besides the issues that can be fixed by correcting staff signatures, orchestral scores can include many other kinds of problematic behaviour that require special attention. For example, 1-line percussive staves like the ones found on the example in Figure 5.18 are not recognized by the OMR software used in this project. Another example are in-line execution hints and alternatives that appear as additional staves inside a grand staff directly above the staff they affect. Such in-line execution hints are recognized by the OMR software as regular staves that are filled by empty bars to span the whole width. Fortunately, both of these issues are not critical to our scenario. The missing events of percussive instruments are not important to our chroma-based comparison of content, and the additional events introduced by in-line execution hints and alternatives are usually very similar to the regular content, so they have no negative effect on the chroma-based comparison either.



## Chapter 6

# Track Segmentation and Identification

For the automatic organization of music documents in a digital music library, one important task is to group all documents that belong to the same piece of music. To this end, one has to choose a level of granularity on which to group the pieces of music. In our scenario, we choose the level of individual movements or songs. The division of higher-level works such as concerts, sonatas, or song cycles into individual movements or songs is uniquely specified by the work entries given in our database of metadata. The work entries are organized in a tree-like hierarchy. We refer to work entries that correspond to higher-level works such as concerts, sonatas, or song cycles as *parent-level works*, and to work entries that correspond to individual movements or songs as *leaf-level works*.

As already discussed in Chapter 4, the music documents that are to be organized are not given as individual songs or movements, but rather as complete sheet music books or audio CD collections that usually contain several leaf-level works or even several parent-level works. The task, now, is to find out which works are contained and to determine the start point and end point of each work in the sheet music books and audio CD collections. In other words, we have to calculate a segmentation of the books and CD collections into segments that correspond to individual works, and for each segment we have to identify the corresponding work entry. We refer to this task as *track segmentation and identification*. A more formal definition of this task including different variants is given in Section 6.1.

In this chapter, we discuss approaches to automatically determine the segmentation and identification. The approaches can be integrated into the document organization system described in Chapter 4. Depending on how robust and reliable the results are, they can be used either for a fully automated organization of the data, or for assisting a user supervising the organization process by getting most of the results right and only requiring some minor corrections. In Section 6.1 we give a more formal definition of the the task of track segmentation and identification including different variants depending on what kind of constraints are given. A straightforward approach to solving the task would be to compare the title headings found in the sheet music documents and CD track titles associated with the CD collections to the

titles of the work entries in the metadata database. In Section 6.2, we present some example data that points out why such an approach based on the comparison of strings is far from trivial and would require research efforts that lie outside the scope of this thesis. Instead, in Section 6.3, we present approaches and experiments towards solving the task through a comparison of musical content, see also [48]. We conclude in Section 6.4 with a discussion on combining the text-based approach and the content-based approach to complement one another and achieve more accurate and robust results.

## 6.1 Definition of the Task

We assume that we are given a sheet music book that contains several leaf-level works. We also refer to a leaf-level work contained in a sheet music book as a *score track*. For the complete sheet music book, we have obtained a set of bar labels  $\mathcal{B}$  with each element of  $\mathcal{B}$  being of the form (page number, line number, bar number) as introduced in Section 3.1, see also Figure 3.2. In the following, we specify locations in the sheet music book via elements of  $\mathcal{B}$ . By sorting the bars in  $\mathcal{B}$  by first page number, then line number, and then bar number, we have obtained a notation bar sequence  $\mu = (\mu_1, \dots, \mu_B), \mu_b \in \mathcal{B}$ . Note, that we, here, use the symbol  $\mu$  instead of the symbol  $\nu$  for the notation bar sequence. The reason for that is that we use the symbol  $\nu$  to indicate a notation bar sequence for a single score track inside a sheet music book. Opposed to that, here, we use the symbol  $\mu$  to denote a notation bar sequence that spans the complete book before it is divided into score tracks.

With these prerequisites, the task of track segmentation and identification for sheet music books can be defined as follows. A set of leaf-level works  $\omega = \{\omega_1, \dots, \omega_Q\}$  is given with  $q \in [1 : Q]$  being called the ID of work  $\omega_q$ . Our given sheet music book contains  $K \leq Q$  of the works in  $\omega$  and can possibly also contain other works that are not part of  $\omega$ . Then, the task of track segmentation and identification is to find  $((s_1, e_1, q_1) \dots, (s_K, e_K, q_K)), (s_k, e_k, q_k) \in [1 : B] \times [1 : B] \times [1 : Q]$  with  $s_1 < \dots < s_K$  such that  $\mu_{s_k}$  is the first bar and  $\mu_{e_k}$  is the last bar in  $\mu$  that belongs to work  $\omega_{q_k}$ . An example is depicted in Figure 6.1. Note that instead of  $(s_k, e_k, q_k)$ , in the figure, the solution is given in the equivalent form  $(\mu_{s_k}, \mu_{e_k}, q_k)$ . Also note that in the example depicted in Figure 6.1, the sheet music book does not contain works that are not part of  $\omega$  that, otherwise, would manifest in sections of the score not being covered by any of the colored boxes.

We distinguish different levels of difficulty for the above task depending on additional constraints that might be given. If the sheet music book is known to contain only works that are part of  $\omega$ , the task becomes easier, because the end points  $e_1, \dots, e_K$  can be determined as

$$\begin{aligned} e_k &= s_{k+1} - 1, k \in [1 : K - 1] \\ e_K &= B. \end{aligned}$$

If, additionally, it is known that all  $Q$  works are contained in the sheet music book, i.e.,  $K = Q$ , the task becomes easier, again, because instead of having to determine the correct subset and permutation of works from  $\omega$ , one only has to determine the correct permutation. In many cases, strong assumptions on this permutation can be made, because the child entries of



**Figure 6.1.** Example for a track segmentation and identification of several pages of a sheet music book containing a total of 4 score tracks. The page numbers are written below the pages. A number on each page indicates to which work the content on the page belongs. Note that on page 16, the upper grand staves belong to work id 2 while the lower grand staves already belong to work id 3.

parent-level works, e.g., the movements of a sonata, usually appear in their canonical order. In case that the canonical order of the works in  $\omega$  is known, and the order of appearance of these works is to be identical in the sheet music book, the task reduces to finding the correct segment boundaries  $s_1 < \dots < s_K$ . In practice, constraints as discussed above can be achieved by using the constraints section of the input mask in the document organization system proposed in Section 4.3, see Figure 4.9.

An analog definition can be made for the task of track segmentation and identification of audio CD collections. Instead of specifying locations via bar labels, here, locations may be specified using a disk number, CD track number, and time position within the CD track. Similar to using a notation bar sequence  $\mu$  to specify the start and end positions through bar numbers  $s_k, e_k \in [1 : B]$ , one may simply count the time from the beginning of the first CD track on the first disk to specify the start and end positions in units of time such as seconds or milliseconds  $s_k, e_k \in \mathbb{R}_{\geq 0}$ .

## 6.2 Text-Based Comparison

Given a sheet music book and a set of work entries  $\omega$  with each work entry specifying some textual information such as a title and composer, an obvious straightforward approach to solve the task of track segmentation and identification is to find title headings in the score and compare them to the work entries. The same could be done for the case of audio CD collections by comparing the information usually found on the backside of CD covers, i.e., titles for each CD track, to the work entries. Even though this might be the obvious approach for human subjects, inferring the correct segmentation and identification based on such a textual comparison is far from trivial to achieve computationally. In this section, we look at several examples of text found in title headings of sheet music, CD track titles, and work entries in our metadata database, and discuss the main difficulties of this approach.

In our examples, we consider three pieces of music: Beethoven’s Piano Sonata No. 1, the “Winterreise” song cycle by Schubert, and the Concert for Violin and Orchestra in E minor, Opus 64, by Mendelssohn. These three pieces of music correspond to parent-level work entries in our metadata database. For each piece, we look at the first movement or song plus one additional movement or song, amounting to total of 6 leaf-level tracks. Excerpts from the sheet music scans corresponding to the beginnings of the score tracks are shown in Figure 6.2. We assume that the text found in the title headings of the sheet music has been classified into different categories according to its position relative to the music staves as shown in Figure 6.3. For the audio CD collections, we use textual information that has been extracted from the Gracenote CD database service. A comparison of the text extracted from the sheet music, text extracted from the Gracenote database for the corresponding audio CDs, and text given for the corresponding work entries in our metadata database is shown in Figures 6.4 to 6.6.

The first example, which is depicted in Figure 6.4, shows the text for the first two movements of Beethoven’s Piano Sonata No. 1, “Allegro” and “Adagio”. The upper section of the figure shows the text extracted from the sheet music classified into the categories introduced in Figure 6.3. There are two different entries including one entry for the first movement and one entry for the second movement. The entry for the first movement contains information regarding the sonata as a whole, e.g. a dedication and an opus number, while the entry for the second movement only contains the tempo directive for the beginning of the movement which simultaneously acts as the title of the movement. The middle section of the figure shows the textual information for the corresponding work entries in our metadata database. There are a total of three work entries involved. The entry listed at the top represents the parent-level work, i.e., the complete sonata, while the other two entries represent the first and the second movement respectively. The hierarchical relationship between the entries is indicated as a tree-like list view in the figure with the two movements being shown as child nodes of the parent-level sonata. Each work entry in the metadata database contains several attributes. These attributes include various different titles following different conventions and standards, a string for work catalog identifiers such as opus numbers, a creation date, and a composer name. Even though the work entries for the individual movements contain the same fields, not all of them are shown in the figure, because they are either empty or identical to the entry of the parent-level work. Note that leaf-level work entries furthermore use an internal attribute that indicates the canonical order of the movements. This allows us to assign numbers to the leaf-level entries as shown in the figure. The lower section of the figure

**SONATE**  
Joseph Haydn gewidmet  
Komponiert 1795  
Opus 2 Nr. 1



**Adagio**  
*dolce*



**Winterreise**  
Wilhelm Müller

**1.**  
Gute Nacht  
Op. 89



**5.**  
Der Lindenbaum



**CONCERT**  
für die Violine mit Begleitung des Orchesters  
von  
**FELIX MENDELSSOHN BARTHOLODY.**  
Op. 64.

Meißeloths Verlag. Serie 4. Nr. 18.

**Allegro molto appassionato.**



**Allegro molto vivace.**



**Figure 6.2.** Example excerpts of sheet music scans showing the title headings for 6 different score tracks. The examples are grouped into three rows. The first row shows headings for the first and the second movement of Beethoven’s Piano Sonata No. 1, Opus 2 No. 1. The second row shows headings for the first and the fifth song of the song cycle “Winterreise” by Schubert. The third row shows headings for the first and the third movement of the Concert for Violin and Orchestra, Opus 64, by Mendelssohn. Note that the headings for the first movement or song can contain information about the parent work, i.e., sonata, song cycle, or concert, while the remaining movements or songs usually do not. The third movement of the Mendelssohn example depicted in the bottom right (titled “Allegro molto vivace”) even starts in the middle of a grand staff.

The figure shows a snippet of a musical score. At the top left, the page number '6' is enclosed in a red box with the label 'Page Number' pointing to it. In the center, the title 'SONATE' is in a large, bold, serif font, with 'Joseph Haydn gewidmet' and 'Komponiert 1795' in smaller text below it, all enclosed in a red box labeled 'Above Center'. To the left of the title, the tempo marking 'Allegro' is in a red box labeled 'Above Left'. To the right, 'Opus 2 Nr. 1' is in a red box labeled 'Above Right'. At the bottom left, the system number '1.' is in a red box labeled 'System Label'. The musical notation consists of two staves with various notes, rests, and dynamic markings like 'p'.


**Figure 6.3.** Illustration of different categories of text related to the title heading of a piece of music in a sheet music book.

shows the textual information available for the corresponding audio CD, which includes fields for artist and title for the CD as a whole plus individual titles for each CD track.

Note that in the illustration in Figure 6.4, the work entries of the metadata database and the entries for the audio CD form some sort of hierarchy but the entries for the text extracted from the sheet music do not. Even though the work hierarchy of sonata and movements manifests itself in different types and amounts of text found for the first and the second movement, we assume that no hierarchy is detected from the sheet music in our examples. Considering that different types of hierarchies exist, e.g. sonata and movement, song cycle and song, or opera, act, and scene, etc., the task of automatically detecting a work hierarchy from the sheet music is not trivial.


The second example is depicted in Figure 6.5 and shows the first and the fifth song from the song cycle “Winterreise” by Franz Schubert. Since in the sheet music book, this song cycle is preceded by another song cycle, the first song of the “Winterreise” cycle has the number 21 in the book. Accordingly, the fifth song has the number 25. In this example, it is interesting to note that different work catalogs are used in the sheet music edition and the metadata of the work entries and the audio CD. The sheet music edition uses an opus identifier “Op.89” while the metadata uses the identifier “D 911” of the so-called “Deutsch-Verzeichnis” which is a special catalog for the works by Franz Schubert initiated by Otto Erich Deutsch. The third example depicted in Figure 6.6 shows the first and the third movement of the Concert for Violin and Orchestra in E minor, Opus 64, by Felix Mendelssohn Bartholdy. Since this is an orchestral piece, the system label of the first grand staff of the score shows the names of all the instrument groups involved in the concert instead of a number for the song or sonata as in the previous examples. Note that in this case, the division of the piece into movements indicated by the CD tracks is different from the reference division specified by the work entries in the metadata database. In the metadata database, the third movement begins with the musical section titled “Allegretto molto vivace” while on the audio CD the third CD track begins with the section “Allegretto non troppo” which is, then, followed by the “Allegro molto vivace”.

From the three examples one can see that finding the correct track segmentation and identification based on a simple comparison of strings extracted from titles and other metadata is not a simple task because of several problems:

 Title headings in sheet music:

	<i>Above Center</i>	<i>Above Right</i>	<i>Above Left</i>	<i>System Label</i>
Movement 1	<b>Sonate</b> Joseph Haydn gewidmet Komponiert 1795	<b>Opus 2 Nr.1</b>	<b>Allegro</b>	<b>1.</b>
Movement 2			<b>Adagio</b>	


 Work entries in metadata database:

	<i>Uniform Title</i>	<i>EST Title</i>	<i>Variant Title</i>
Parent	Drei Klaviersonaten (f-moll, A-dur, C-dur) - I.	Sonaten, Kl, op. 2, 1	Klaviersonate Nr. 1 f-moll op. 2 Nr. 1
	<i>Opus</i>	<i>Creation Date</i>	<i>Composer Name</i>
	op. 2, 1	1795	Beethoven, Ludwig van

	<i>Uniform Title</i>	<i>EST Title</i>
Leaf 1	<b>Allegro</b>	Sonaten, Kl, op. 2, 1 <1. Satz>
Leaf 2	<b>Adagio</b>	Sonaten, Kl, op. 2, 1 <2. Satz>

 Audio CD information from Gracenote:

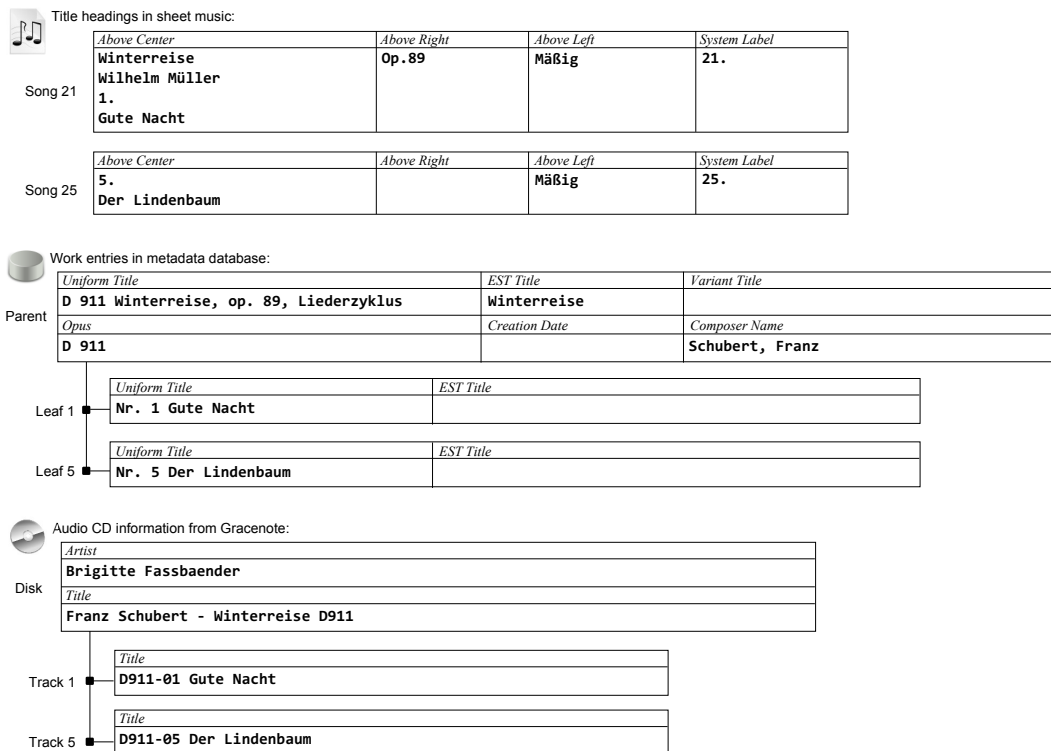
	<i>Artist</i>
Disk	Ludwig Van Beethoven, Alfred Brendel
	<i>Title</i>
	Complete Piano Sonatas, Disc 1 - Alfred Brendel - Philips Classics

	<i>Title</i>
Track 1	Sonata no.1 in F minor, op.2 no.1 / Allegro
Track 2	Sonata no.1 in F minor, op.2 no.1 / Adagio

**Figure 6.4.** Comparison of textual metadata for the first and second movement of Beethoven’s Piano Sonata No. 1. The top part shows text found in the title headings of the sheet music, the middle part shows text specified in the corresponding work entries in the metadata database, and the bottom part shows textual metadata extracted from the Gracenote database for the corresponding audio CD.

1. For each entry, we do not just have a single string, but possible many strings of different categories that might or might not be filled with content. It is not clear which strings should be compared to which other strings.
2. In the sheet music and in the Gracenote data for the audio CDs, there is no reliable convention for which information is written to which field. For example, in the case of Beethoven and Schubert, the work catalog identifier (opus number) is found at the “Above Right” position, but in the case of Mendelssohn it is found at the “Above Center” position. Also at the “Above Center” position, the Mendelssohn example contains the name of the composer. However, in the other two examples, the name of the composer is not included. Instead, the Beethoven example includes the name “Joseph Haydn” (who is a composer himself but not the composer of this piece) as a dedication. The Schubert example includes the name “Wilhelm Müller”, who is the originator of the lyrics but not the composer of the music. In the Gracenote data for the audio CDs, in the Beethoven example, the composer name is part of the disk artist field. In the Schubert example, on the contrary, the composer name is part of the disk title.
3. Various languages and formats might be used, e.g., “f-moll” vs. “F minor”, “Concert für die Violine” vs. “Violin Concerto”, or “op. 2, 1 “ vs. “op.2 no.1”.
4. Different work catalogs might be used causing differences in identifiers such as opus numbers.




**Figure 6.5.** Comparison of textual metadata for the first and fifth song of the “Winterreise” cycle by Franz Schubert.

- The division of a parent-level work into leaf-level works specified by the work entries in the metadata database might be different from the division of content into CD tracks on the audio CD.
- The text elements that represent title headings have to be found in the sheet music. In many cases this can be supported by searching for indented grand staves indicating the beginning of a new movement. However, in general, this criterion is not guaranteed to hold, as, for example, is the case in the Mendelssohn example where the transitions between the movements are seamless in the sheet music.
- The results of the text recognition from the sheet music scans might contain errors.


Considering the above issues, it seems clear that a successful approach to track segmentation and identification based on the comparison of the textual entries cannot simply perform a fault-tolerant matching of strings and substrings. It rather seems necessary to make heavy use of additional prior knowledge about the meaning of certain names, key words, and expressions found in the textual entries. Using a database of composer names, commonly used work types and titles, and work catalogs including different styles of writing, abbreviations, and different languages, one might be able to detect and interpret the key components of the textual entries. After having determined the composer and type of work, e.g., sonata, song, concert, or opera, one may use further knowledge for determining possible work catalog identifiers and other key components that are typical for the given composer and work type.



 Title headings in sheet music:

	<i>Above Center</i>	<i>Above Right</i>	<i>Above Left</i>	<i>System Label</i>
Movement 1	<b>CONCERT für die Violine mit Begleitung des Orchesters von FELIX MENDELSSOHN BARTHOLDY Op. 64.</b>		<b>Allegro molto appassionato.</b>	<b>Flauti. Oboi. Clarineti in A. Fagotti. ...</b>
Movement 3			<b>Allegretto molto vivace.</b>	


 Work entries in metadata database:

	<i>Uniform Title</i>	<i>EST Title</i>	<i>Variant Title</i>
Parent	<b>Concert für die Violine</b>		
	<i>Opus</i>	<i>Creation Date</i>	<i>Composer Name</i>
	<b>op. 64</b>		<b>Mendelssohn Bartholdy, Felix</b>

	<i>Uniform Title</i>	<i>EST Title</i>
Leaf 1	<b>Allegro molto appassionato</b>	
Leaf 3	<b>Allegretto molto vivace</b>	

 Audio CD information from Gracenote:

	<i>Artist</i>
Disk	<b>Viktoria Mullova, ASMF, Sir Neville Marriner</b>
	<i>Title</i>
	<b>Mendelssohn Violin Concertos - Viktoria Mullova, ASMF, Sir Neville Marriner</b>

	<i>Title</i>
Track 1	<b>Mendelssohn Violin Concerto in E minor, Op.64 - I. Allegro molto appassionato</b>
Track 3	<b>Mendelssohn Violin Concerto in E minor, Op.64 - III. Allegretto non troppo - Allegro molto vivace</b>

**Figure 6.6.** Comparison of textual metadata for the first and third movement of the Concert for Violin and Orchestra in E minor, Opus 64, by Felix Mendelssohn Bartholdy.

The detected key components can then be converted to a standardized form and can then be suitably compared.

Even though pursuing this approach would be very interesting for this project, we leave this text-based comparison open for future work. Following the global direction of this thesis, we, instead, focus on an approach based on the comparison of musical content, i.e., a comparison based on note events extracted from the score and the acoustic content of the given audio recordings. In the following Section 6.3, we present experiments towards answering the question how well the task of track segmentation and identification can be solved by the comparison of musical content only. If for a work entry in the metadata database some musical content has already been identified previously, a content-based comparison can be used to complement the text-based comparison of title headings to lead to an overall improved robustness in the segmentation and, especially, the identification of new content.

As a final note in this section, instead of trying to detect title headings near the beginnings of score tracks in the sheet music, a different strategy would be to search the beginning of the sheet music book for a table of contents that might include useful information for the segmentation and identification in a quite compact form. In case that the table of contents list all movements together with their full name and work catalog number and their corresponding page number in the sheet music book, one could use this information to identify the leaf-level works contained in the book and to find the corresponding start pages by identifying the page numbers printed on the sheets. Note that these page numbers printed on the sheets may

**BAND I**

Seite

Seite

1. Allegro Opus 2 Nr. 1 6

2. Allegro vivace Opus 2 Nr. 2 22

3. Allegro con brio Opus 2 Nr. 3 45

4. Allegro molto e con brio Opus 7 71

5. Allegro molto e con brio Opus 10 Nr. 1 96

6. Allegro Opus 10 Nr. 2 110

7. Presto Opus 10 Nr. 3 124

8. Grave Opus 13 (Pathétique) 146

9. Allegro Opus 14 Nr. 1 164

10. Allegro Opus 14 Nr. 2 177

11. Allegro con brio Opus 22 193

12. Andante con Variazioni Opus 26 216

13. Andante Opus 27 Nr. 1 234

14. Adagio sostenuto Opus 27 Nr. 2 249

15. Allegro Opus 28 263

**Figure 6.7.** Example for a table of contents found in the sheet music book of the Beethoven Piano Sonatas Volume 1 published by G. Henle.

be different from the page numbers used in our set of bar labels  $\mathcal{B}$ , because usually not all pages at the beginning of the books are counted in their printed numbering of pages. The major drawback of this approach is that the types and forms of table of contents can vary to a high degree between different types of sheet music, publishers, and editions. In many cases, incipits of the beginning of the score are printed with only sparse additional textual information, see Figure 6.7 for an example. Automatically extracting information out of such table of contents that can assist in the task of track segmentation and identification certainly poses its own area of research that cannot be pursued as part of this thesis.

### 6.3 Content-Based Comparison

In this section, we assume that a set of works  $\omega = \{\omega_1, \dots, \omega_Q\}$  is given and that we already have organized some audio CD collections such that for each work  $\omega_q$ ,  $q \in [1 : Q]$  we have the audio data  $\alpha_q$  of a corresponding audio recording available. Given a sheet music book, we want to perform track segmentation and identification as defined in Section 6.1 to add the sheet music to the repository of our digital music library. Since, at this point, musical content is available for each work  $\omega_q$  through the audio data  $\alpha_q$ , the segmentation and identification of the content of the sheet music book can be performed by comparing it to this already known and organized audio content.

The content-based comparison is done as follows. First, in a preprocessing step, all audio recordings are converted to sequences of chroma vectors, see Section 3.4. While keeping book on document boundaries, all these chroma sequences are concatenated into a single audio feature sequence  $y = (y_1, \dots, y_M)$ . Keeping book on the document boundaries allows us to determine the corresponding work ID  $q \in [1 : Q]$  for any given index position  $m \in [1 : M]$ . We write this as a mapping  $f : m \mapsto q$ . Then, the note events extracted from the sheet music scans are also converted to a single sequence of chroma features. From this sequence, segments of different lengths are drawn and are subsequently used as queries to be compared to the audio feature sequence. A single query sequence  $x = (x_1, \dots, x_N)$  might, for example, correspond to a single page of sheet music or a certain sequence of grand staves. The query feature sequence  $x$  is compared to the audio feature sequence  $y$  using subsequence dynamic time warping (SSDTW) as described in Section 3.5. In our experiments, it turned out that the SSDTW step sizes  $(2, 1)$ ,  $(1, 2)$ ,  $(1, 1)$  (instead of the classical step sizes  $(1, 0)$ ,  $(0, 1)$ ,  $(1, 1)$ ) lead to more robust matching results, and are hence used in the remainder of this section. As a result of the SSDTW computation, we obtain a set of matches  $H(x, y)$ . Recall that the elements of  $H(x, y)$  are alignment paths of the form  $p = (p_1, \dots, p_L)$  with  $p_\ell = (n_\ell, m_\ell) \in Z = [1 : N] \times [1 : M]$  for  $\ell \in [1 : L]$ . Let  $p \in H(x, y)$  be the match with minimum cost  $c(p) = D(n_L, m_L) = D(N, m_L) = \sum_{\ell=1}^L C(n_\ell, m_\ell)$ . We also refer to  $p$  as the *top match*. From the end position  $m_L$  of the top match in the audio feature sequence, we determine the corresponding work number  $q \in [1 : Q]$  by making use of the previously stored audio document boundaries through  $q = f(m_L)$ . We, then, consider the query segment of sheet music as being identified as part of work  $\omega_q$ . The idea of the content-based approach to track segmentation and identification, now, is to choose suitable queries from the sheet music data and use the SSDTW method described above to determine the boundaries and work IDs of the score tracks.

Experiments have been conducted on the basis of the 32 piano sonatas by Ludwig van Beethoven. Because of its outstanding musical significance and the large number of available digitized audio recordings, the automated analysis and organization of the corpus of Beethoven's piano sonatas is highly relevant to musicologists and librarians. The audio database consists of a complete recording of the 32 piano sonatas conducted by Daniel Barenboim, which includes  $Q = 101$  individual movements with a total duration of 11 hours. As for the sheet music book, we use a scanned version of the G. Henle edition. Even though this edition originally comes as two separate volumes, we consider the scans as a single sheet music book containing a total of 604 pages showing a total of 3693 two-stave grand staves, in the following also referred to as *lines*.

The scanned pages have been processed by the OMR Engine of SharpEye 2.68 as described in Section 2.3. However, since the experiments have been conducted in an earlier stage of the project, the conversion of the OMR output to note events, i.e., note onsets, pitches, and durations, has been performed differently from the approach described in Chapter 5. Instead of parsing the MRO files output by the SharpEye software directly, the output was first converted to MusicXML using the internal export function of SharpEye. The resulting MusicXML files were, then, parsed and converted to chroma features with a resolution of 1 Hz. For the conversion, a constant tempo of 100 BPM was assumed. No postprocessing was used to correct obvious errors such as mismatching key signatures in the staves for the left and right hand. Furthermore, the settings used in the conversion simply discarded any

voices whose accumulated duration did not match the duration of the detected time signature leading to missing note events. Since the errors in the key signatures and the missing notes certainly affect the content-based comparison via chroma features in a negative way, the rates achieved with OMR data in the experiments below might improve when using a more accurate way of deriving note events.

### 6.3.1 Baseline Experiment

In a baseline experiment, we investigated what identification rates one may expect in the case that there are no severe OMR extraction errors, that the jumps and repeats in the score have been detected correctly, and that the approximate tempo of each movement is known. To this end, we used a complete set of MIDI files for the 32 Beethoven sonatas and randomly generated a large number of MIDI fragments of various lengths, which were used instead of the OMR extraction results. Then, for each of these MIDI fragments we computed the top match with respect to the audio database. Recall that in the identification scenario the objective is to determine the piece of music underlying the respective MIDI fragment by using the audio recordings of the database as an identifier. Therefore, we consider a match as *correct* if it lies within the audio document that corresponds to the same movement as the MIDI document from which the respective query is taken. Otherwise the match is considered as *incorrect*.

In particular, we investigated the dependency of the number of correct audio matches subject to the length  $A$  (given in seconds) of the MIDI query. To this end, we randomly generated 1000 MIDI queries for each of the seven parameters  $A \in \{10, 20, 30, \dots, 70\}$ . Each of the queries lies within a single MIDI file and therefore has a unique correct assignment to one of the 101 movements. The second column of Table 6.1 shows the number of correct matches. As an example, consider the case  $A = 10$ , where 823 of the 1000 matches were correct. Note that the number of correct matches increases significantly with the query length. For example, for  $A = 40$  only 3 of the 1000 queries were misclassified. To give a more detailed picture of the matching quality, Table 6.1 additionally provides various cost and confidence values. The third, fourth, and fifth column show the average cost values, the standard deviations, and the maximal cost values for the correct top matches. For example, in the case  $A = 10$ , the average cost value (standard deviation/maximal cost value) for the 823 correct matches is 0.059 (0.024/0.223). The latter cost values are with respect to a range from 0 (no costs) to 1 (maximum costs). Increasing  $A$  leads to slightly higher cost values stabilizing around the value 0.07 even for long queries.

Similarly, the sixth, seventh, and eighth columns of Table 6.1 show the corresponding values for the incorrect top matches. For example, in the case  $A = 10$ , the average cost of the 177 incorrect top matches is 0.084 with a standard deviation of 0.034. Note that in the case of incorrect matches, when increasing the query length, the average cost increases at a much higher rate than in the case of correct matches.

We also investigated how well the correct matches were separated by successive matches that do not lie in the respective correct audio document. To this end, we computed for each query the minimal cost value of a restricted matching curve, where the correct audio document had been removed. Then, for all correctly identified queries, we computed the difference of this

Length (in sec.)	#(Cor.) (in %)	Cost (correct)			Cost (incorrect)			Gap av.
		av.	std.	max.	av.	std.	max.	
10	82.3	0.059	0.024	0.223	0.084	0.034	0.207	0.044
20	96.7	0.068	0.026	0.206	0.102	0.031	0.196	0.070
30	99.2	0.070	0.024	0.189	0.139	0.040	0.214	0.093
40	99.7	0.071	0.024	0.218	0.177	0.027	0.198	0.106
50	99.9	0.072	0.023	0.204	0.117	0.000	0.117	0.118
60	99.9	0.071	0.021	0.193	0.159	0.000	0.159	0.128
70	99.9	0.071	0.022	0.196	0.229	0.000	0.229	0.135

**Table 6.1.** Results for the baseline experiment of mapping MIDI fragments of various lengths  $A \in \{10, 20, \dots, 70\}$  (given in seconds) to the audio database. Each line shows the length  $A$ , the percentage of correct matches for the 1000 MIDI fragments of the respective length, the average values (av.), the standard deviations (std.), and the maximum values (max.) of the correct matches and incorrect matches, and the average confidence gap.

minimal value and the cost of the correct top match. This difference value, which we refer to as *confidence gap*, indicates the identification reliability based on the top match. The average confidence value is shown in the last column of Table 6.1. For example, in the case  $A = 10$  the average confidence gap amounts to the value 0.044. Increasing  $A$  leads to a significant increase of the confidence gap up to the value of 0.135 for  $A = 70$ . In conclusion, one may say that one obtains very good identification rates (with an error rate of less than 1%) for MIDI fragments of at least 30 seconds of duration.

### 6.3.2 Real-World Experiment

Next, we describe a similar experiment, now using the (potentially flawed) OMR extraction results instead of the “clean” MIDI data. For each of the 604 scanned pages, we computed a chroma feature sequence as explained in Section 6.3. Then, from these sequences, we randomly generated 1000 subsequences of length  $A$  for each of the length parameters  $A \in \{10, 20, \dots, 70\}$ . Table 6.2 summarizes the OMR-audio mapping results. Obviously, the identification rate drops significantly compared to the pure MIDI case. For example, in the case  $A = 10$  only 484 out of the 1000 OMR query fragments appear as top match in the correct audio document (opposed to the 823 correct matches in the MIDI case). The identification rate increases to roughly 87% for OMR feature sequences that correspond to a duration of 50 seconds and above. A comparison with Table 6.1 shows that, in the OMR case, the average costs of the correct matches are much higher than the ones in the MIDI case. Furthermore, the confidence gap is much smaller. All these numbers indicate that the OMR-audio mapping procedure significantly suffers from the artifacts that are caused by OMR extraction errors, the missing tempo information, and the lossy conversion to of OMR data to note events.

We continue the analysis of our OMR-audio mapping procedure based on the raw OMR material. Instead of using randomly chosen OMR fragments of a specific duration, we now investigate the mapping quality based on musical units such as pages or lines. Using entire pages in the OMR-audio mapping leads to an identification rate of roughly 82.5%. The average length of the corresponding chroma sequences amounts to 55 seconds yielding robust mappings if there are no severe OMR errors. Another problem that often leads to misclassifications is

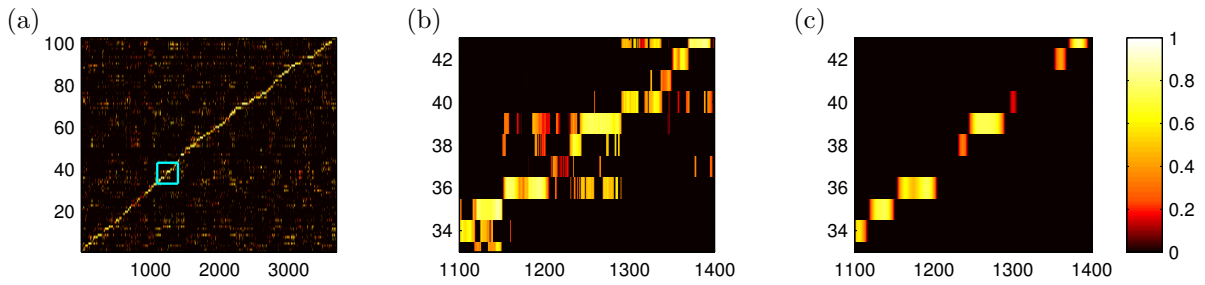
Length (in sec.)	#(Cor.) (in %)	Cost (correct)			Cost (incorrect)			Gap av.
		av.	std.	max.	av.	std.	max.	
10	48.4	0.080	0.033	0.198	0.104	0.040	0.247	0.034
20	67.9	0.103	0.039	0.261	0.147	0.051	0.285	0.050
30	78.4	0.114	0.044	0.292	0.173	0.049	0.317	0.062
40	84.9	0.120	0.043	0.356	0.192	0.051	0.340	0.072
50	87.1	0.132	0.043	0.305	0.208	0.051	0.367	0.080
60	87.0	0.143	0.050	0.304	0.232	0.044	0.356	0.080
70	87.1	0.153	0.052	0.316	0.247	0.049	0.373	0.078

**Table 6.2.** Experimental results mapping OMR fragments of various lengths (given in seconds) to the audio database. For each length parameter  $A \in \{10, 20, \dots, 70\}$  we randomly generated 1000 OMR chroma subsequences, each corresponding to a subpart of exactly one of the scanned pages. The table has the same interpretation as Table 6.1.

Lines	Length (in sec.)	$k = 1$	$k = 2$	$k = 5$	$k = 10$	$k = 20$	$k = 50$
		(in %)	(in %)	(in %)	(in %)	(in %)	(in %)
1	9.133	44.57	52.97	65.77	76.20	84.59	92.26
3	27.099	71.30	76.66	83.62	88.06	92.45	96.13
5	45.053	77.04	81.23	86.41	90.12	93.37	96.86
7	62.995	77.74	81.83	86.84	90.85	93.83	96.89

**Table 6.3.** Identification rates depending on the number of lines used in the OMR-audio mapping. The columns indicate the recall percentage (out of 3693 mappings, respectively) of the correct audio document within the top  $k$  matches.

that a single scanned page may refer to more than one pieces of music. In particular for our Beethoven corpus, a single page may contain both the end and the beginning of two consecutive movements as for example on page 16 in Figure 6.1. To overcome this problem, one may use single lines in the mapping process instead of entire pages. This also yields the advantage of having several identifiers per page. On the downside, the average length of the chroma sequences corresponding to the lines lies below a duration of 10 seconds yielding an identification rate of only 44.57%, see Table 6.3. To improve the identification rate of the line-based mapping strategy, we query each line in the context of  $a$  preceding and  $a$  subsequent lines. In other words, instead of using a single line we use a block of  $2a + 1$  subsequent lines with the reference line positioned in the middle. Here, we assume that all pages belonging to one movement are in the correct order, hence allowing us to consider blocks of lines ranging across two consecutive pages. To systematically investigate the identification rate depending on the number of lines used in the OMR-audio mapping, for each of the 3693 lines of our scanned Beethoven material, we generated chroma query sequences corresponding to 1, 3, 5, and 7 lines. Table 6.3 shows both the resulting identification rates based on the top match ( $k = 1$ ) and the recall values for the correct audio document for the top  $k$  matches with  $k \in \{1, 2, 5, 10, 20, 50\}$ . For example, using three lines, the top match ( $k = 1$ ) was correct in 71.30% of the 3693 OMR-audio mappings. Considering the top 5 matches ( $k = 5$ ), at least one of these matches was correct in 83.62% of the mappings.



**Figure 6.8.** (a) Mapping matrix  $W$  for the Beethoven scenario. The rows correspond to the audio documents ( $Q = 101$ ) and the columns to the OMR queries ( $U = 3693$ ). (b) Enlargement of the marked region of  $W$ . (c) The same region after applying the postprocessing procedure.

### 6.3.3 Postprocessing Clean-Up

We now show how the additional information of considering the  $k$  top matches (instead of considering only the top match) can be used to detect most of the incorrect identifications. The only assumption we use is that the scanned pages that correspond to a specific movement are given as a sequence of consecutive pages, i.e., pages of different movements are not interleaved. We explain our postprocessing procedure by means of our Beethoven scenario using  $U = 3693$  OMR queries each consisting of 7 subsequent lines and considering the  $k = 5$  top matches. Recall that the objective is to map each of the queries to one of the  $Q = 101$  audio documents (representing the pieces or movements). We construct a  $Q \times U$  *mapping matrix*  $W$ , where the rows correspond to the pieces and the columns to the queries. Then an entry  $W(q, u)$ ,  $1 \leq q \leq Q$ ,  $1 \leq u \leq U$ , is non-zero if and only if the  $q$ -th audio document appears among the top  $k$  matches for the  $u$ -th query. In this case  $W(q, u)$  is set to  $1 - c$ , where  $c \in [0, 1]$  denotes the cost of the corresponding match. In case there are several matches for the entry  $(q, u)$  among the top  $k$  matches, we define  $c$  to be the minimal cost value over these matches. Note that  $W(q, u)$  expresses a kind of confidence that the  $u$ -th query belongs to the  $q$ -th piece. Furthermore,  $W$  indicates the kind of confusion that occurred in the identification procedure. Fig. 6.8 shows the mapping matrix for the Beethoven scenario.

For our Beethoven corpus, both the audio recordings and the scanned pages are sorted with respect to increasing opus and movement numbers. Therefore, a correct mapping of all queries corresponds to a diagonal staircase-like structure in  $W$ . In the following, we do not assume that the scanned pages are given in the same order (on the piece and movement level) as the audio recordings, since this assumption is often violated in real-world digitization applications. For example, many music books contain a more or less unsorted mixture of various pieces and movements. Therefore, we only make the assumption that the pages that correspond to a specific audio document (referring to a specific movement) are given in the correct order. Then, in case of a correct identification of the OMR queries, the matrix  $W$  reveals a structure of horizontal line segments, where each such segment corresponds to an audio document.

In the following, a tuple  $(q, u)$  is referred to as *positive* if the entry  $W(q, u)$  is non-zero. Furthermore, a positive tuple  $(q, u)$  is referred to as *true positive* if the  $u$ -th query semantically corresponds to the  $q$ -th audio document, otherwise  $(q, u)$  is called *false positive*. Now, the idea is that positive tuples included in long horizontal line segments within  $W$  are likely to be true, whereas isolated positive tuples are likely to be false. Intuitively, our procedure classifies

the positive tuples by looking for groups of tuples included in long horizontal line segments (these tuples are classified as true) and discards isolated positive tuples (these tuples are classified as false). We refer to Figure 6.8 for an illustration.

We have applied this postprocessing procedure to the Beethoven scenario using  $U = 3693$  queries each consisting of 7 subsequent lines and considering the top match only. As a result, 78.42% of the queries were mapped correctly and 17.17% of the queries were not mapped (by discarding false positives). The remaining 4.41% are incorrect mappings. Note that the result of this type of postprocessing is the detection rather than the correction of incorrect identifications. Having identified incorrect mappings allows to both further improve the identification process and to automatically reveal passages within the sheet music where severe OMR errors have occurred.

Rather than identifying incorrect mappings, one may also increase the number of correct identifications. For this, certain tuples are specified as true positives by “filling” small gaps within horizontal line segments. Thus, OMR queries are assigned to a specific audio document if neighboring OMR queries are consistently assigned to the same audio document. Using  $k = 3$  in our example increases the number of correct identifications to 86.70% (instead of 77.74% without postprocessing). Note that there is a natural trade-off between eliminating the incorrect identifications and boosting the correct identifications.

## 6.4 Conclusions

Obtaining both accurate and reliable results in computationally solving the task of track segmentation and identification seems to be problematic when using either a text-based comparison of titles and metadata or a content-based comparison alone. The text-based approach alone can deliver precise locations of possible segment boundaries but is not very well suited for identifying the segments with high certainty. The content-based approach alone can identify longer segments with high certainty but is not very well suited for delivering precise segment boundaries. Obviously, the two approaches complement each other very well. The discussions and experiments of this chapter suggest that by combining the two approaches, good overall results may be achieved in future automatic approaches to track segmentation and identification. Such an approach can be easily integrated into the document organization system described in Chapter 4. In case that, for a given document, a track segmentation and identification of the content is determined with high certainty, no further user interaction might be required. Cases with only low certainty could be reported to the user and, if necessary, manually corrected.



## Chapter 7

# Partial Synchronization

In this chapter we discuss approaches to computing a sheet music-audio synchronization in the presence of structural differences. We also refer to this task as *partial synchronization*. We assume that we are given a score track and an audio track for which is known that they represent the same leaf-level work and that their track boundaries are correct. Furthermore, we assume that we have an estimate  $\delta^*$  of the default bar sequence  $\delta$  (see Table 4.1) available. The structural differences can either originate from COMA cases (see Section 4.4), differences between the default bar sequence  $\delta$  and the performance bar sequence  $\pi$ , or from inaccuracies in our estimation  $\delta^*$  (see Section 5.4).

The general idea followed in this chapter is to exploit structural elements found in the sheet music, e.g., double bar lines, to divide the sheet music data into blocks such that structural differences between the sheet music and the audio data only happen between blocks, but never inside blocks. This reduces the task of handling general differences in structure to finding a sequence of blocks that matches the performance bar sequence. We discuss three approaches to finding the correct sequence of blocks, all of which are based on content-based comparison.

First, in Section 7.1, we identify possible origins of structural differences between score tracks and corresponding audio recordings. In Section 7.2, we explain how score tracks may be divided into blocks such that no jumps are expected to occur within these blocks. Sections 7.3 to 7.5 introduce three approaches to determining the sequence of blocks that best resembles the performance in the audio recording, all of which rely on content-based comparison. Section 7.3 discusses an approach that uses the matching techniques based on subsequence dynamic time-warping similar to the approach for track segmentation and mapping described in Section 6.3. The main idea of the second approach, which is presented in Section 7.4, is to start with an initial block sequence and iteratively apply classical dynamic time-warping to infer modifications to the block sequence by detecting degenerations in the resulting alignment path. Section 7.5 covers the third approach, which proposes an extension of the dynamic time warping algorithm to allow taking into account possible repeats and jumps between the individual blocks of the score data. The first two approaches have been studied as part of a diploma thesis project [53]. The third approach is a further development that eliminates most of the disadvantages of the previous two approaches. Therefore, the main focus of this chapter is set on the approach of Section 7.5 and the previous two approaches are only summarized

briefly. Several additional variants, as well as experiments and results of this approach are presented in the Subsections 7.5.1 to 7.5.6.

## 7.1 Types of Structural Differences

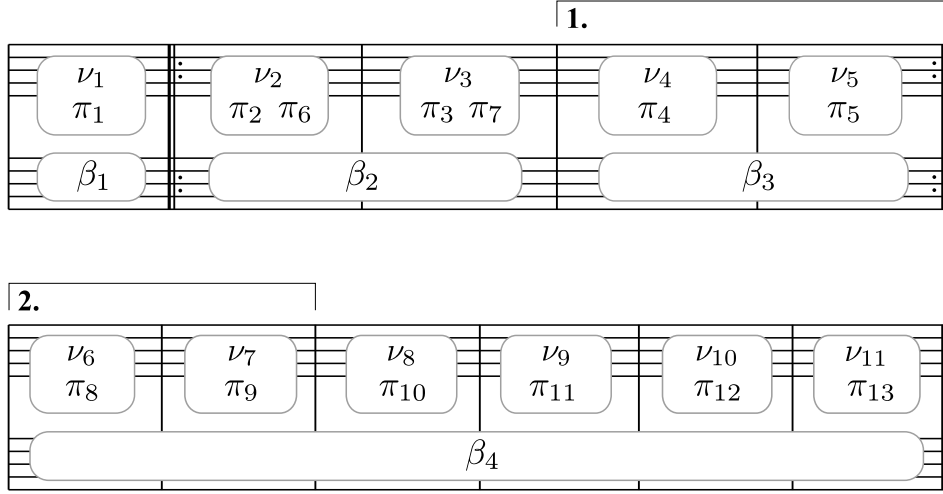
We are given a score track and an audio track that may exhibit differences in their global structure. The sheet music that constitutes the score track consists of a set of bars with notation bar sequence  $\nu$  as introduced in Chapter 3. By detecting repeats and jumps from the sheet music as accurately as possible, we have derived an estimate  $\delta^*$  of the default bar sequence  $\delta$ , see Section 5.4. The bar sequence played in the performance given in the audio track is denoted as  $\pi$ . Note that in our scenario,  $\pi$  is unknown. For an overview of the different bar sequences, see Figure 3.2 and Table 4.1. Structural differences between the score track and the audio track manifest as differences between the estimated default bar sequence  $\delta^*$  and the performance bar sequence  $\pi$ . There are two causes for such structural differences:

- (SD1) Inaccuracies in our estimation  $\delta^*$ , i.e., differences between  $\delta^*$  and  $\delta$ . In our scenario, such differences may be caused by complex jump directives not being recognized correctly from the sheet music, as well as by repeats with alternative endings, which are not recognized by the OMR software used. Multi-repeats, as often found in songs with more than two verses, are not recognized as well. Another cause for differences between  $\delta^*$  and  $\delta$  could be missed out repeat signs, but since the recognition of repeat signs is very reliable, we expect the amount of these cases to be small.
- (SD2) Differences between the default bar sequence  $\delta$  and the performance bar sequence  $\pi$ . Note that such difference can be either due to COMA cases, as have been discussed in Section 4.4, or by performers choosing to deliberately ignore or add repeats and jumps. Furthermore, the performance might be based on a different sheet music edition which uses a different default bar sequence.

In our scenario, we assume that the COMA cases are known through manually created COMA entries in the PROBADO Document Organization System as illustrated in Figure 4.12. In the following sections of this chapter, we discuss strategies for handling the remaining structural differences by computing estimates for the unknown performance bar sequence  $\pi$ . Assuming that the resulting estimates are accurate, the sheet music-audio synchronization can be computed by following the methods introduced in Chapter 3 using the estimate of  $\pi$  instead of  $\delta$ .

## 7.2 Jumps and Blocks

Compared to audio data, sheet music data offers simple access to a lot of structural information. One important aspect of this structural information, which we will make use of in the following, is that the notation bar sequence  $\nu = (\nu_1, \dots, \nu_B), \nu_b \in \mathcal{B}$  of a score track can



**Figure 7.1.** Illustration of the notation bar sequence  $\nu$ , the performance bar sequence  $\pi$ , and the score block sequence  $\beta$ . In this example, the block boundaries are  $b_1 = 1, b_2 = 3, b_3 = 5, b_4 = 13$ .

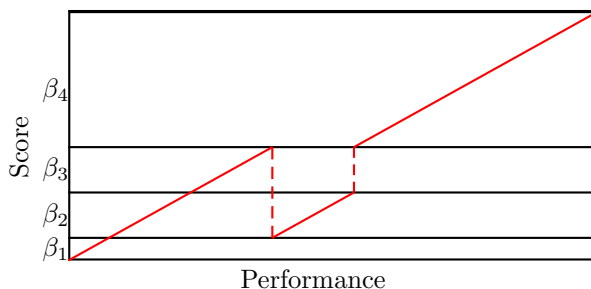
usually be segmented into a small number of subsequences of bars such that no repeats or jumps are expected to occur in the performance within these subsequences. In the following, we will specify more formally what we mean by that.

First, let us recall that the performance bar sequence  $\pi = (\pi_1, \dots, \pi_K)$ ,  $\pi_k \in \mathcal{B} \cup \{\uparrow\}$  consists of elements that are either bar labels pointing to particular bars, or the element  $\uparrow$  indicating a section that is not written in the sheet music. A *jump* at position  $k$  in the performance bar sequence  $\pi$  with *source bar*  $\nu_s$  and *target bar*  $\nu_t$  means that  $\pi_k = \nu_s \Rightarrow \pi_{k+1} = \nu_t$  with  $t \neq s + 1$ , i.e., the performance bar sequence does not follow the notation bar sequence at this position. We refer to this type of jump as an *internal jump*. Besides such jumps within the score where both the source bar and the target bar are elements of  $\mathcal{B}$ , a different kind of jump is present if either  $\pi_k = \uparrow$  or  $\pi_{k+1} = \uparrow$ . These are the cases, where the performance jumps from the sheet music to a section that is not written in the sheet music, or back into the sheet music after having performed a section that is not written in the sheet music. We refer to this type of jump as an *external jump*. Considering an internal jump with source bar  $\nu_s$  and target bar  $\nu_t$ , we speak of a *backward jump* if  $t < s$ , and of *forward jump* if  $t > s + 1$ . Repeats are simply a special type of backward jump which we will not further distinguish in the following.

Let  $0 < b_1 < \dots < b_{I-1} < B$  be the indices of the bars in  $(\nu_1, \dots, \nu_{B-1})$  that act as either a jump source or that are immediately followed by a bar that acts as a jump target. Adding  $b_0 = 0$  and  $b_I = B$ , we define the *block*

$$\beta_i = (\nu_{b_{i-1}+1}, \dots, \nu_{b_i}) \quad (7.1)$$

of length  $|\beta_i| = b_i - b_{i-1}$  for  $i \in [1 : I]$ . The resulting *score block sequence*  $\beta := (\beta_1, \dots, \beta_I)$  is a partition of  $\nu$ . We also refer to the indices  $b_1, \dots, b_I$  as the *block boundaries*. See Figure 7.1 for an example. Now, the task of finding the performance bar sequence  $\pi$  is reduced to finding a sequence of block indices  $g = (g_1, \dots, g_J)$ ,  $g_j \in [1 : I], j \in [1 : J]$  such that  $(\beta_{g_1}, \dots, \beta_{g_J})$  resembles the performance as closely as possible. We refer to the sequence

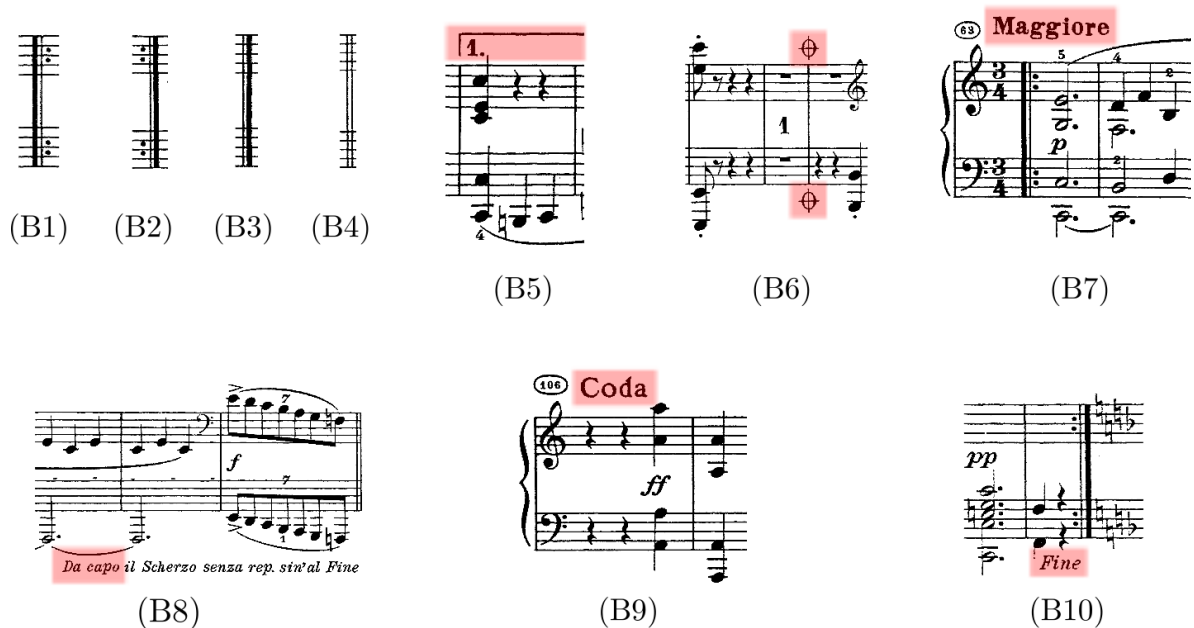


**Figure 7.2.** Visualization of a score-audio synchronization result with performance block index sequence  $g = (1, 2, 3, 2, 4)$  for the notation bar sequence and performance bar sequence shown in Figure 7.1. The dashed parts of the red line indicate jumps.

$g$  as the *performance block index sequence*. For an example, we refer to Figure 7.2. For approaches based on content-based comparison, finding a block sequence  $(\beta_{g_1}, \dots, \beta_{g_I})$  that matches the performance may be considered easier than finding the more general performance bar sequence  $\pi$  directly, because the average block length is usually much bigger than 1 (a single bar) and the amount of blocks  $I$  is usually much smaller than the amount of bars  $B$ .

The block boundaries  $b_1, \dots, b_I$  can be obtained as structural information from the sheet music by searching the score data for so-called *block boundary indicators*, see Figure 7.3. Alternatively, they can also be derived from a given default bar sequence  $\delta$  or estimate  $\delta^*$ , because the jump indicators they are derived from are block boundary indicators as well. However, opposed to having to derive complete jumps specified as source-target pairs, as is the case for determining  $\delta^*$ , for the block boundaries, single indicators that might suggest either a jump source or a jump target are sufficient. This allows us to include block boundary indicators for which no complete jump can be derived, e.g., an isolated segno marker for which we could not identify a matching source or target to pair up with. Furthermore, we can include indicators that suggest jumps in the performance only with a relatively high uncertainty. Such indicators are for example double bar lines without repeat signs, obstructed text elements that might originally contain keywords such as “da capo”, “segno”, or “coda”, and text elements that might represent title headings indicating the beginning of a new musical section. Not all occurrences of these indicators necessarily indicate a source or target for a jump that is actually made in the performance bar sequence. However, the only drawback we get from including an unnecessary block boundary is that a block is divided into two smaller blocks. If the two smaller blocks are still long enough, a possible negative effect on the content-based comparison might be practically irrelevant. In practice, we prefer to include some unnecessary block boundaries over missing out a block boundary, because a missing block boundary means that the performance bar sequence  $\pi$  can no longer be perfectly matched by any block sequence  $\beta$ .

In our real-world scenario, we have the unfavorable situation that some types of block boundary indicators are not recognized at all, namely jump marker symbols such as  $\%$  or  $\oplus$  and horizontal brackets for repeats with alternative endings. With no evidence for these indicators available, the respective block boundaries cannot be found. Candidates for block boundaries according to repeats with alternative endings may be guessed in the presence of double bar lines with closing repeat signs. This approach will be discussed briefly in Section 7.5. Better

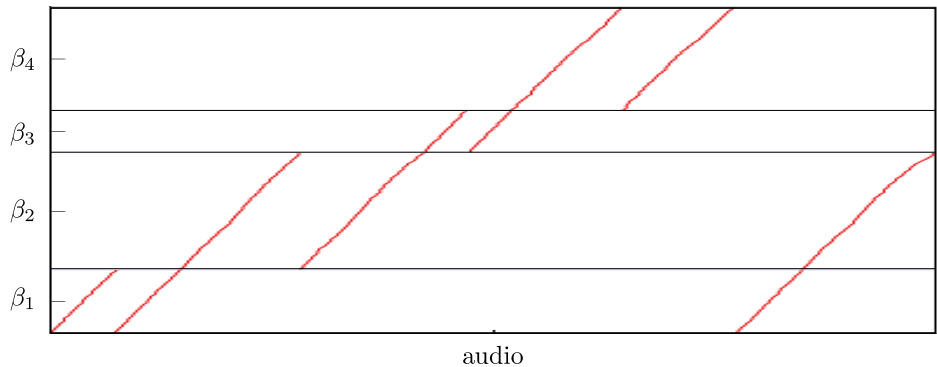


**Figure 7.3.** Examples for several types of block boundary indicators. (B1) double bar line with opening repeat signs, (B2) double bar line with closing repeat signs, (B3) double bar line of type thin-thick, (B4) double bar line of type thin-thin, (B5) beginning of horizontal bracket for a repeat with alternative endings, (B6) jump marker (segno), (B7) text element indicating beginning of a new musical section, (B8) keyword *da capo*, (B9) keyword *coda*, (B10) keyword *fine*.

results could, however, be achieved by incorporating OMR software that successfully recognizes and outputs these missing indicators.

Assuming that the block boundaries  $b_1, \dots, b_I$  are determined and the corresponding partition of  $\nu$  into blocks  $\beta = (\beta_1, \dots, \beta_I)$  is derived, the remaining task is to find the performance block index sequence  $g = (g_1, \dots, g_J)$ . We can make use of the fact that sheet music usually follows certain rules regarding the arrangement of blocks to create constraints for the sequence  $g$ . Here are some example constraints that could be applied. The particular set of rules depends on the type of piece and instrumentation.

- There is a maximum of 1 jump induced by a *da capo*.
- A block cannot be repeated more than  $k \in \mathbb{N}$  times in a row.
- After a backward jump to a block  $\beta_i$ , one cannot perform backward jumps to blocks  $\beta_j$  with  $j < i$  unless that jump is induced by a *da capo*.
- After reaching a block  $\beta_i$  that starts with an opening repeat sign, one cannot perform backward jumps to blocks  $\beta_j$  with  $j < i$  unless that jump is induced by a *da capo*.
- A forward jump can only be induced by a beginning of horizontal bracket for a repeat with alternative endings or by a jump marker (segno).
- After a *da capo* jump, blocks cannot be repeated.



**Figure 7.4.** Visualization of perfect matching results for a piece with four blocks  $\beta = (\beta_1, \beta_2, \beta_3, \beta_4)$  and the performance block index sequence  $g = (1, 1, 2, 2, 3, 3, 4, 4, 1, 2)$ .

Such rules can be used to narrow down the possible choices in the computation of  $g$  discussed in the following sections. Furthermore, a given default bar sequence  $\delta$  or an estimate  $\delta^*$  may also be used as additional information. Since the performance is expected to follow the default bar sequence  $\delta$  in most cases, a given  $\delta$  or  $\delta^*$  can be used to derive an *expected performance block index sequence*  $g^*$ , which will be used in some of the approaches for determining  $g$  discussed in the remaining sections of this chapter.

### 7.3 Matching Approach

Given the sequence of blocks  $\beta$ , the main idea of this approach is to use content-based queries to identify where and how often each block  $\beta_i, i \in [1 : I]$  occurs in the audio data. To this end, the subsequence dynamic time-warping technique as described in Section 3.5 can be used. The content to search in is the audio data of the given audio track. To this end, a feature sequence  $y$  is created from the audio data. For each block  $\beta_i \in \beta$ , we compute the corresponding feature sequence  $x^i$ , which we also refer to as *feature blocks*. Each feature block  $x^i$  is queried one after another, leading to a set of matches  $H(\beta_i) := H(x^i, y) = \{p^{1,i}, \dots, p^{R_i,i}\}$  for each queried block, with  $R_i$  being the amount of matches obtained for block  $\beta_i$ . Each match  $p^{r,i}, r \in [1 : R_i], i \in [1 : I]$  delivers a temporal region  $T(p^{r,i}) = [s(p^{r,i}), t(p^{r,i})]$  in the audio track as well as a cost  $c(p^{r,i})$  that indicates how certain it is that the match is correct, see Section 3.5 for details.

The resulting matches can nicely be visualized in two dimensions, see Figure 7.4. The horizontal axis measures time in the audio track. The vertical axis measures the position in the notation bar sequence  $\nu$  in the score track and shows the segmentation of  $\nu$  into blocks  $\beta_i$ . Each match  $p^{r,i}$ , i.e., the  $r$ -th match for the  $i$ -th block, is visualized as a path starting at the beginning of  $\beta_i$  on the vertical axis and time position  $s(p^{r,i})$  on the horizontal axis and ending at the end of  $\beta_i$  and time position  $t(p^{r,i})$ . Actually, this way of visualization can be imagined as having a complete local cost matrix for comparing the score track with the audio track and then plotting paths that are found by performing subsequence DTW on horizontal slices of that cost matrix.

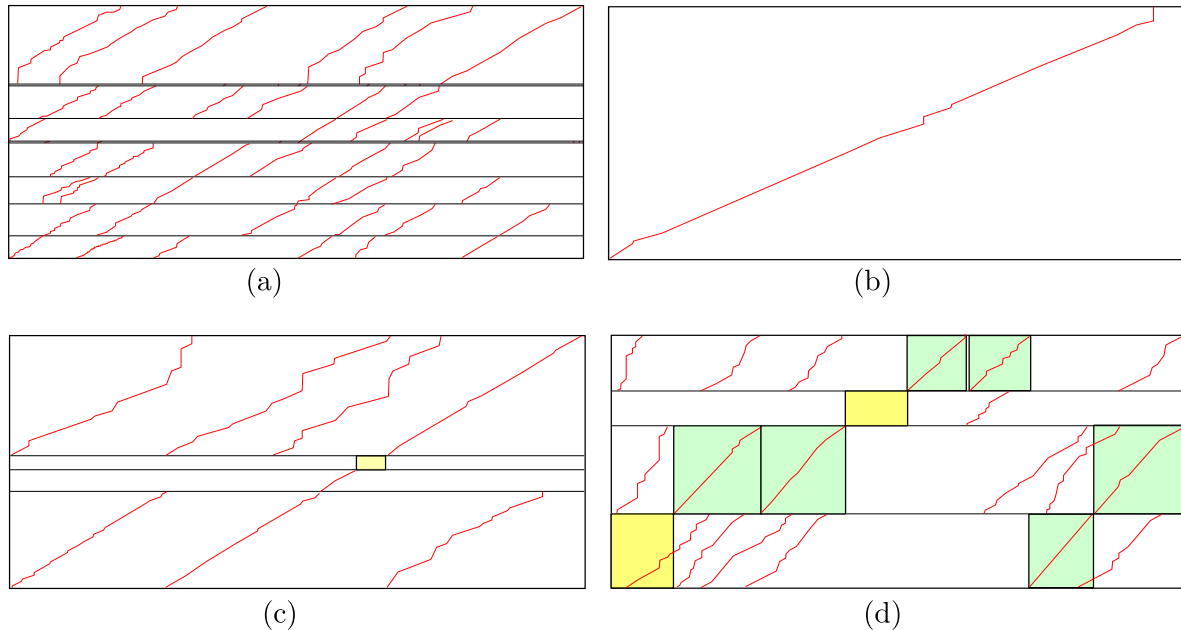
Having the set  $H := \bigcup_i H(\beta_i)$  of all matches for all the blocks available, the performance block index sequence  $g$  can be computed as the block indices corresponding to the most likely sequence of matches  $h = (p_1, \dots, p_J), p_j \in H$  to cover the complete audio track. What is the most likely sequence is determined by the following criteria.

- (C1) The end time of the temporal region in the audio of match  $p_j$  should be close to the start time in the audio of match  $p_{j+1}$ , i.e.,  $|t(p_j) - s(p_{j+1})|$  should be small for all  $j \in [1 : J - 1]$ .
- (C2) The sum of the cost of the matches used in  $h$ , i.e.,  $\sum_{j=1}^J c(p_j)$ , should be small.
- (C3) The sequence should meet a given set of constraints such as proposed at the end of Section 7.2.

Even though this straightforward approach seems promising, the experiments in [48, 45, 53] have revealed several problems and issues. First of all, the matches returned by subsequence DTW are not very reliable, especially for queries that are short. However, many pieces contain a lot of short blocks that lead to erroneous matches, especially in the presence of OMR errors. Furthermore, if different blocks are similar in their musical content, subsequence DTW will not be able to distinguish the occurrences of these blocks in the audio recording, see Figure 7.5(a). Even though, in general, the subsequence DTW is capable of dealing with differences in tempo, the matching results are still sensitive to major differences of factor 2 or higher [45]. Furthermore, the particular course of the path of each match  $p_j$  becomes less reliable towards its start and end position, which leads to inaccuracies in the start times  $s(p_j)$  and end times  $t(p_j)$ , see Figure 7.5(b). These inaccuracies, then, have a negative influence on criterion (C1).

In some cases it can happen that none of the generated matches cover a certain section of the audio track. Such cases may be caused by OMR errors obstructing the correct match. The gap in the coverage of the audio track can lead to confusion in the algorithm for finding the most likely sequence of matches  $h$  to cover the complete audio track. For example, wrong matches might be preferred over correct ones to keep the gap as small as possible. To avoid this, one could allow gaps in the coverage and fill up the gaps at a later time by “inventing” matches that are reasonable by some criteria. Several examples of problematic cases for the matching approach are illustrated in Figure 7.5.

Even though the approach of computing  $g$  via content-based search using subsequence DTW might deliver correct results in more simple cases where the block size is large, the robustness of the approach is low for pieces that comprise many smaller blocks. Getting lower robustness with decreasing block size has to be expected, since it is a general property of content-based approaches. However, in this particular approach, extra instability is introduced by splitting the task of computing a globally optimal  $g$  into two stages. In the first stage, the matches are not constrained or guided by the global optimization criteria that are relevant in the second stage. Without this guidance from the global criteria, the matching leads to mistakes and confusion that cannot be fixed or compensated in the second stage anymore.



**Figure 7.5.** Illustration of several problematic cases for the matching approach. a) Blocks with similar content, b) match with inaccurate end time, the correct match should end at the upper right corner of the box, c) missing match (yellow box), d) two missing matches (yellow boxes), correct matches are indicated by green boxes.

## 7.4 Path Degeneration Approach

After having learned from the matching approach that querying each block individually introduces additional confusion, the approach described in this section tries to deal with the task of computing the performance block index sequence  $g$  more globally. The *path degeneration approach* makes use of the observation that when performing a global synchronization via DTW, differences in structure usually lead to detectable degenerations in the alignment path. Starting with some initial block index sequence  $g^0 = (g_1^0, \dots, g_{j_0}^0)$  for the score track, the idea is to alter the block index sequence trying to eliminate the degenerations. This is done as follows. For each block  $\beta_i \in \beta$ , we compute the corresponding feature block  $x^i$ . Then, we create a feature sequence  $x$  for the whole score track by concatenating the feature blocks according to the initial block sequence  $g^0$ , i.e.,  $x = \bigcup_{j=1}^{j_0} x^{g_j^0}$  where  $\bigcup$  is to be understood as the concatenation operation. This feature sequence is used together with the feature sequence  $y$  created from the audio track to perform DTW and obtain an alignment path. Based on degenerations found in the alignment path, we make alterations to the block sequence. Then, based on the altered block sequence, we create a new feature sequence for the score track and perform DTW again to check if the alterations have eliminated the degenerations.

Roughly speaking, a *degeneration* is a section of the alignment path exceeding a particular minimum length that is completely horizontal or vertical. This would mean that while time moves forward in one of the sequences, it stands still in the other sequence. If there are no differences in structure, such a situation is not expected to happen, except for the very



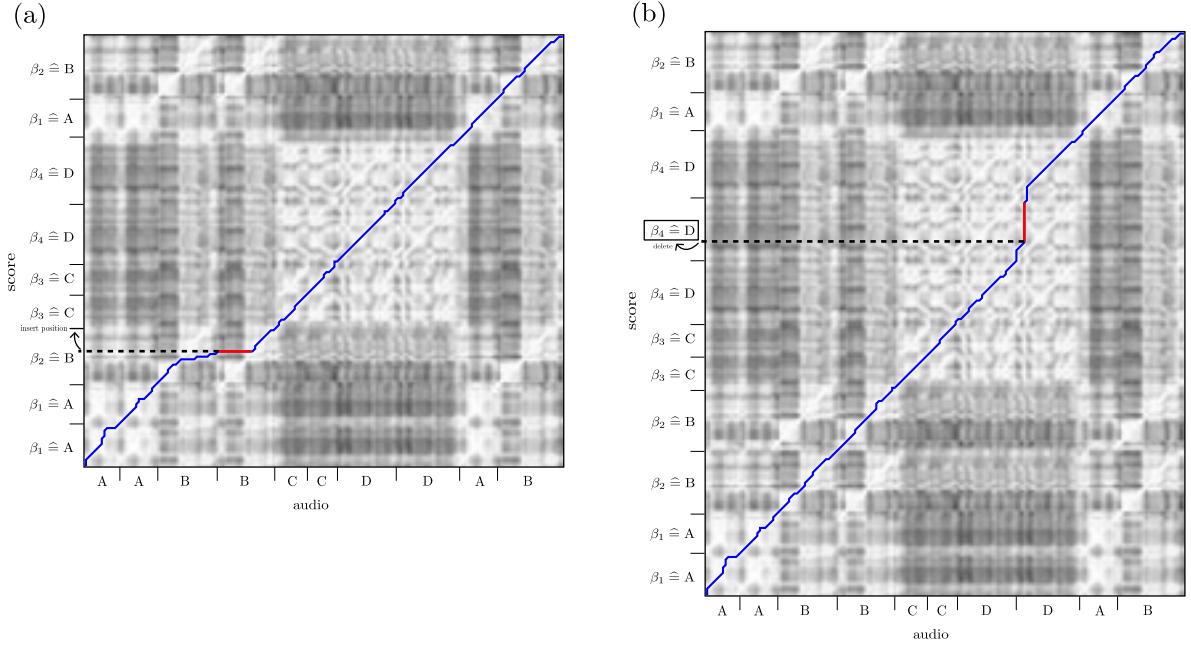
beginning and end of the path, where additional silence in the audio recording can be present. In this approach, we assume that the silence at the beginning and end of the audio recording have been stripped in a preprocessing step. Therefore, the presence of degenerations can be used as an indicator for structural differences. Degenerations can be assigned properties, e.g., orientation (horizontal or vertical), position, length, and cost. These properties can be used to estimate in what way the block sequence of the score track must be altered to eliminate the degeneration. Possible alterations are the insertion of a block into the sequence at a particular position, or the deletion of a block from the sequence. In the following, we call such an alteration an *action*.

The properties of degenerations indicate which actions might eliminate the degeneration. In the following, we define what we mean by degenerations and their properties more formally. Given an alignment path  $p = (p_1, \dots, p_L)$  with  $p_\ell = (n_\ell, m_\ell)$  as defined in Section 3.1, and a threshold  $\theta \in \mathbb{N}_{>0}$ , a subsequence  $f = (p_a, p_{a+1}, \dots, p_{a+h-1}) \subseteq p$  with  $h \geq \theta$  is called a *horizontal degeneration* if  $n_a = n_{a+1} = \dots = n_{a+h-1}$ . It is called a *vertical degeneration* if  $m_a = m_{a+1} = \dots = m_{a+h-1}$ . In both cases,  $a$  is called the *start position*,  $a + h - 1$  is called the *end position* and  $h$  is called the *length* of the degeneration. The *cost* is defined as the sum of local cost along the subsection of the path, i.e.,  $c(f) = \sum_{\ell=a}^{a+h-1} C(n_\ell, m_\ell)$  with  $C$  being the local cost matrix.

For each degeneration, we can create a list of possible actions that might eliminate the cause of the degeneration. This list is created as follows. For a horizontal degeneration, we assume that the cause is a missing block in the vertical sequence, i.e., the feature sequence of the score data. In that case, we determine the index in the block sequence where the missing block should be inserted. This is done by taking the vertical component  $m_a$  of the start position  $p_a$  of the degeneration, and finding the index of the block in the concatenated feature sequence whose start position in the feature sequence is closest to  $m_a$ . In case  $m_a$  is closer to the end of the sequence than to the start of the last block, the target insert position is at the end of the sequence. Then, for each available block that could be inserted, we add an action to the list that inserts the block at that position. Note, that the amount of blocks that are eligible for insertion can be reduced by taking into account a given set of constraints for the block sequence as proposed at the end of Section 7.2. For a vertical degeneration, we assume that the cause is an extra block in the sequence of score data that is not played in the audio recording. In that case, we only add a single action to the list, which is to remove the block at the corresponding position. See Figure 7.6 for an example of both a horizontal and a vertical degeneration and how the insert/remove position for possible actions to fix the degeneration are determined.

Given a score track and an audio track with differences in structure, the path degeneration approach works iteratively using the following greedy strategy. We use the index  $k \in \mathbb{N}_0$  as a superscript in parantheses to count the iterations.

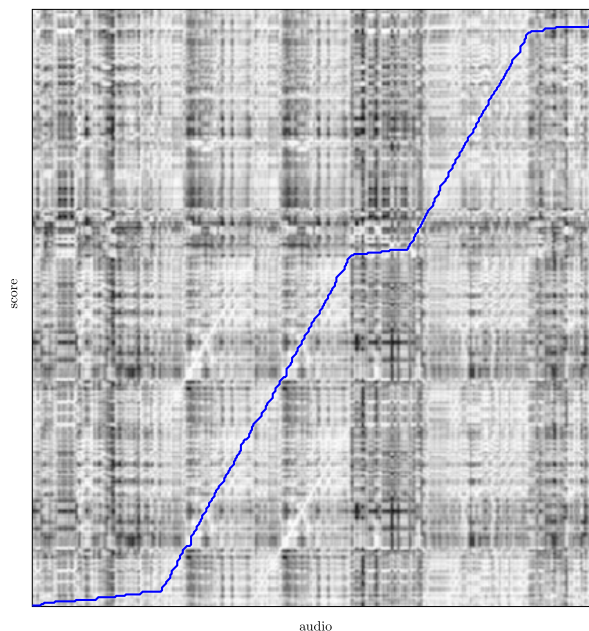
1. Compute the feature sequence  $y$  for the audio track. Compute the feature blocks  $x^i$  for each block  $\beta_i$ ,  $i \in [1 : I]$  of the score track.
2. Set the iteration counter  $k = 0$ . Set the initial block index sequence  $g^{(k)} = g^{(0)}$  to the block index sequence  $g^*$  that corresponds to the estimated default bar sequence  $\delta^*$ .



**Figure 7.6.** Examples for (a) horizontal degeneration and (b) vertical degeneration. In both cases, the audio recording has a block structure AABBCDDAB with A corresponding to score block  $\beta_1$ , B to  $\beta_2$ , C to  $\beta_3$ , and D to  $\beta_4$ . In case (a), a  $\beta_2$  block is missing, which causes a horizontal degeneration. The insert position for the missing block is derived from the vertical position of the degeneration. In case (b), an extra  $\beta_4$  block causes a vertical degeneration. Based on the vertical start position of the degeneration, the middle one of the three subsequent  $\beta_4$  blocks is selected for deletion.

Concatenate the feature blocks  $x^i$  according to  $g^{(k)}$  to obtain the concatenated score feature sequence  $x^{(k)}$ .

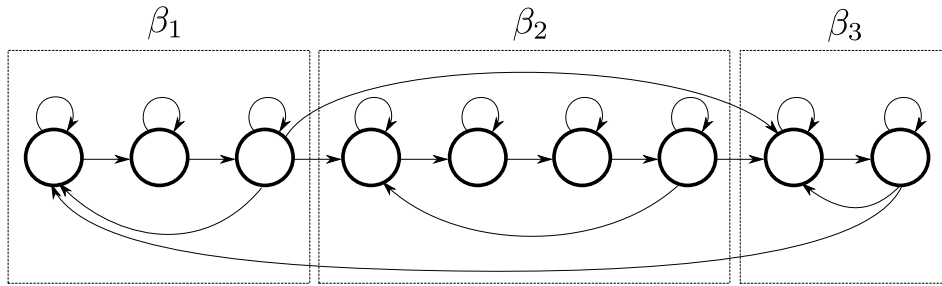
3. Perform DTW as described in Section 3.2 to obtain an alignment path  $p^{(k)}$ .
4. Scan the path for degenerations and obtain the set of degenerations after  $k$  iterations  $D^{(k)}$ .
5. If no degeneration is found, i.e., if  $|D^{(k)}| = 0$ , terminate the algorithm and use the current block sequence as the result  $g = g^{(k)}$ . Otherwise, select the degeneration  $d^{(k)} \in D^{(k)}$  with the smallest start position and generate the list  $E^{(k)}$  of possible actions on  $g^{(k)}$  for trying to eliminate  $d^{(k)}$ .
6. If the list  $E^{(k)}$  is not empty, perform and remove the first action from the list by inserting or removing the corresponding block index into/from  $g^{(k)}$  to create  $g^{(k+1)}$ . Create the concatenated score feature sequence  $x^{(k+1)}$  based on  $g^{(k+1)}$  and continue with step 7. Otherwise, if the list  $E^{(k)}$  is empty, it means that the degeneration  $d^{(k)}$  could not be fixed. If  $k > 0$ , roll back to iteration  $(k - 1)$  and continue with step 6. Otherwise, if  $k = 0$  terminate the algorithm with an error message.
7. Perform DTW again and obtain a path  $p^{(k+1)}$  as well as a corresponding set of degenerations  $D^{(k+1)}$ .



**Figure 7.7.** An alignment path that has degenerations even though the block index sequence  $g$  for the score data is correct. The illustration is based on the score and a performance of the first movement of Sonata 8, Op. 13, “Pathétique” by Beethoven. The degenerations are caused by large variations in tempo throughout the performance of the piece that are not taken into account in the score data. Throughout the piece, slow sections that are performed with as little as 20 BPM alternate with fast sections that are performed with as high as 300 BPM. However, the score data uses an estimated mean tempo of 172 BPM. The large tempo difference between the slow sections of the performance and the estimated mean tempo cause the appearance of horizontal degenerations in the alignment path even though the path is actually correct.

8. Check if the degeneration  $d^{(k)}$  has been fixed. The degeneration  $d^{(k)}$  is considered fixed if the smallest start position of any degeneration  $d \in D^{(k+1)}$  is bigger than the end position of  $d^{(k)}$ . If the degeneration has been fixed, increase the iteration counter  $k$  and continue with step 5. Otherwise, discard  $g^{(k+1)}$ ,  $x^{(k+1)}$ ,  $p^{(k+1)}$ , and  $D^{(k+1)}$ , and proceed with step 6.

Compared to the matching approach discussed in the previous section, the path degeneration approach makes better use of the global optimization criteria by trying to identify the problems in the context of a global synchronization. Experiments have shown, that the approach has indeed better robustness than the matching approach [53]. A major drawback is, however, that computing a global synchronization for each test in each iteration is computationally expensive. For pieces with many blocks, the runtime of the algorithm can become problematic, see [53] for details. A weak point in the accuracy of the approach is the assumption that degenerations in the alignment path always indicate a missing or extra block in the block sequence of the score. It turns out that degenerations can also be caused by major differences between the estimated tempo of the score and the actual tempo of the audio recording. Especially for pieces that comprise significant tempo changes within the piece or even within blocks, degenerations tend to appear even if the block sequence is correct, see Figure 7.7.



**Figure 7.8.** HMM for a score track with three blocks  $\beta = (\beta_1, \beta_2, \beta_3)$ . Each block is represented by a left-to-right connected sequence of states. Repeats and jumps between blocks are simply modeled through extra connections from the last state of a block to the first state of another block.

Another limiting factor of the approach might be that in each iteration, only one action can be performed at a time. Theoretically, it could happen that, to reach the correct block sequence, more than one action must be taken in one step to fix a degeneration appearing in the alignment path. In such a case, the approach described above would fail and output an error message. Such cases might, however, not occur very often in practice.

## 7.5 JumpDTW

In Chapter 3 we have introduced two approaches based on dynamic programming, namely dynamic time warping (DTW) and hidden markov models (HMM), for computing an alignment path for two sequences that are assumed to have no differences in structure. In this section, we propose an extension to the DTW approach that integrates an advantage of the HMM approach to form an algorithm that outputs an optimum alignment path in presence of differences in structure. The main idea of this approach is to allow the alignment path to make jumps within the local cost matrix instead of restricting it to follow a fixed set of step vectors only. Accordingly, we refer to this extended algorithm as *JumpDTW*.

The idea of allowing jumps in the alignment path is inspired by the way a piece of music is modeled with an HMM. Let us assume, we are given a score track that consists of a sequence of blocks  $\beta = (\beta_1, \dots, \beta_I)$ . Each block  $\beta_i, i \in [1 : I]$  is modeled as a sequence of left-to-right connected states. The left-to-right style of connection models the behaviour that the position in the score can only move forward but never backward. In this model, it is straightforward to account for possible repeats and jumps by simply adding further connections that connect states representing possible jump sources to states representing possible jump targets, see Figure 7.8 for an example.

This idea can be translated to the DTW approach by extending the concept of an alignment path and the DTW algorithm as follows. Recall that we assume that the jumps occur from ends to beginnings of the blocks  $\beta_i, i \in [1 : I]$ . With regard to the feature sequence  $x = (x_1, \dots, x_N)$  of the score, we assume that the beginning of  $\beta_i$  corresponds to index  $s_i \in [1 : N]$  and the end to index  $t_i \in [1 : N]$ , where  $s_i < t_i$ . Note that the beginning of block  $\beta_{i+1}$  immediately follows the end of block  $\beta_i$ , i.e.,  $s_{i+1} = t_i + 1$ . Let  $S := \{s_i \mid i \in [1 : I]\}$  and

$T := \{t_i \mid i \in [1 : I]\}$ . An *alignment path with jumps* with respect to the sets  $S$  and  $T$  is defined to be a sequence  $p = (p_1, \dots, p_L)$  with  $p_\ell = (n_\ell, m_\ell) \in Z$  for  $\ell \in [1 : L]$  satisfying the boundary condition as before (P1). However, this time we modify the step condition (P2) by requiring that either  $p_\ell - p_{\ell-1} \in \Sigma$  (as before) or

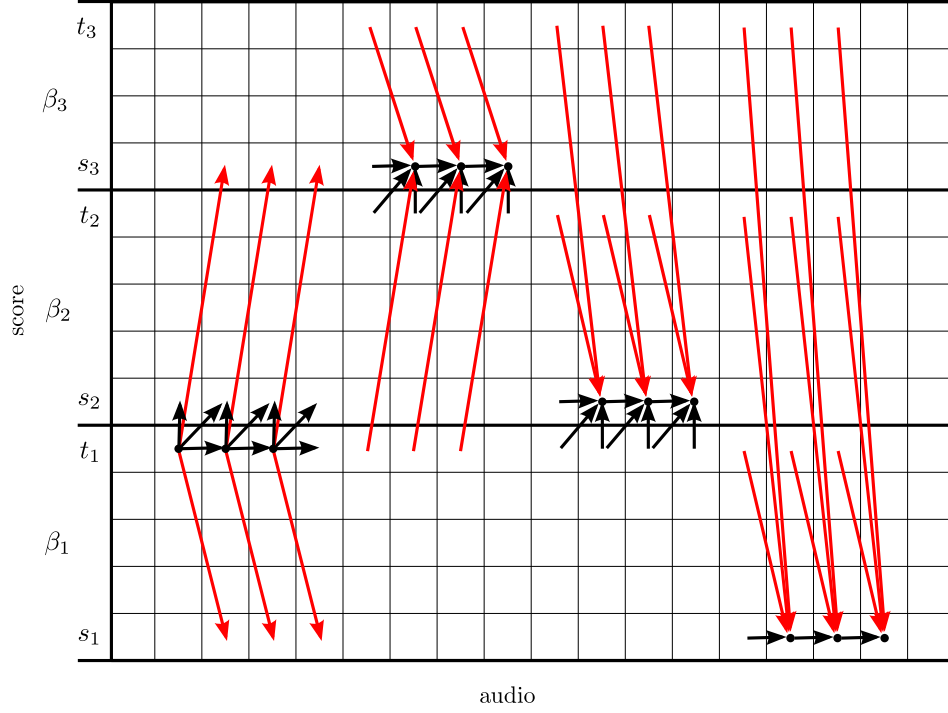
$$m_{\ell-1} = m_\ell - 1 \wedge n_{\ell-1} \in T \wedge n_\ell \in S. \quad (7.2)$$

In other words, besides the regular steps, we also permit jumps in the first coordinate (corresponding to the score) from the end of any block (given by  $T$ ) to the beginning of any other block (given by  $S$ ). Note that by allowing jumps only in one of the two input sequences we switch from the symmetric behavior of the DTW approach to an asymmetric behavior in the way the input sequences are treated. Recall, that this asymmetric behavior is also a property of the HMM approach, as discussed briefly in Section 3.3.

We now introduce a modified DTW version, referred to as *JumpDTW*, that allows for computing an optimal alignment path with jumps. In classical DTW, when calculating the accumulated cost for a given cell of the accumulated cost matrix  $D$ , we only consider cells as possible predecessors in the path that are connected to the current cell through the fixed set of step vectors  $\Sigma$ . In the classical setting, one chooses  $\Sigma = \{(0, 1), (1, 0), (1, 1)\}$  meaning that each cell can be reached from its left, bottom, and bottom-left neighbor. In Equation 3.6, we achieved that by maintaining for each cell  $(n, m)$  a set  $Z_{n,m}$  of possible predecessor positions from which cell  $(n, m)$  can be reached with a single step vector  $\zeta \in \Sigma$ . In fact, we have chosen this way of describing the classic DTW algorithm so that our following extension can be made with little effort. For convenience, from now on, we refer to entries in the sets  $Z_{n,m}$  of possible predecessor positions as *connections* with an entry  $(n', m') \in Z_{n,m}$  being referred to as a connection from  $(n', m')$  to  $(n, m)$ . We, furthermore, call the connections that are used in the classic DTW algorithm *regular connections*. The main idea of our extension, then, is that we can easily model jumps in the input sequences that are to be synchronized by adding further connections.

Note, that, opposed to the HMM concept depicted in Figure 7.8 where jumps can only be modeled in one of the sequences, in the case of our modified DTW approach, we can theoretically model jumps in both of the sequences. However, allowing both forward and backward jumps for both of the sequences, we can run into situations, where we get possible loops in the alignment path, which lead to recursion in the calculation of the accumulated cost matrix, compare for Equation 3.7. To effectively avoid such situations, we allow backward jumps only for one of the sequences, which, in our case, we choose to be the vertical sequence representing the score track. Furthermore we require backward jumps in the score sequence to move forward in the audio sequence by at least by one frame. Using these restrictions, the calculation of the accumulated cost can be performed column-wise from left to right by iterating from bottom to top within each column and is guaranteed to be free of recursions.

To calculate the performance block index sequence  $g$  using JumpDTW, we perform the following steps. We calculate the feature sequence for the score track according to the notation bar sequence  $\nu$ . This means, the feature sequence resembles the sequence of blocks  $\beta$  without taking any repeats or jumps into account. For each block  $\beta_i \in \beta$ , the first index  $s_i$  in the feature sequence and the last index  $t_i$  in the feature sequence are known. After having calculated the local cost matrix and having initialized the connection sets  $Z_{n,m}$  according to



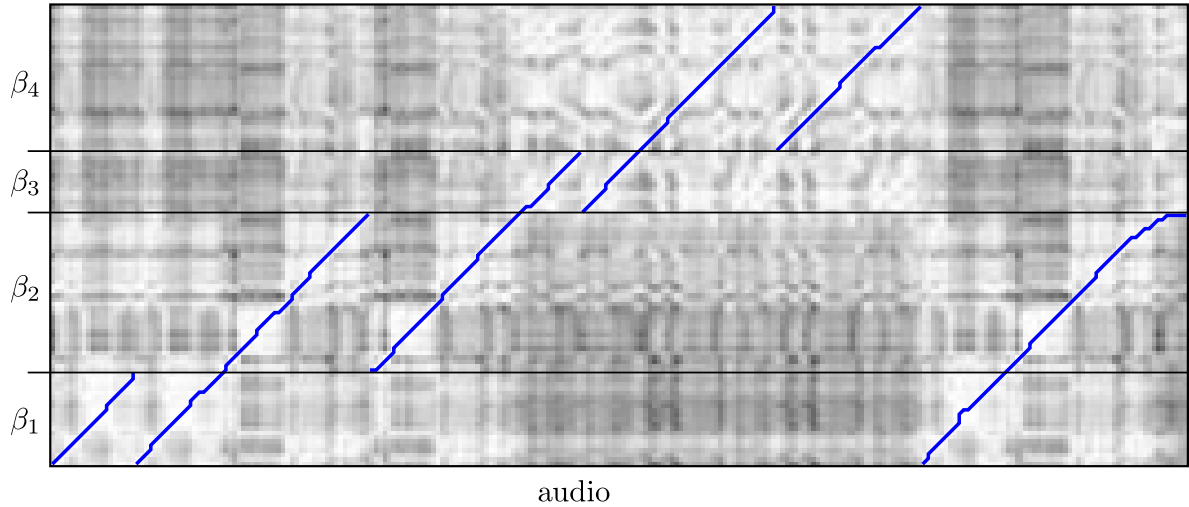
**Figure 7.9.** Illustration of cost matrix with regular connections (black arrows) and jump connections (red arrows). In this example, the score track consists of three score blocks  $\beta = (\beta_1, \beta_2, \beta_3)$ . The first and last index positions in the score feature sequence of each block  $\beta_i$  are denoted  $s_i$  and  $t_i$  respectively. Except for cells at the matrix borders, each cell has three outgoing regular connections. In addition to that, for each column in the rows  $t_1$ ,  $t_2$ , and  $t_3$ , we add special connections to the cells in rows  $s_1$ ,  $s_2$ , and  $s_3$  of next column that are not yet reachable through a regular connection. This is indicated for the case of  $t_1$  in the illustration by the leftmost group of arrows. Instead of outgoing connections, equivalently, in our algorithm we save incoming connections for each cell  $(n, m)$  in the sets  $\tilde{Z}_{n,m}$ . Except for cells at the matrix borders, each cell can be reached through three incoming regular connections. Furthermore, the cells in rows  $s_1$ ,  $s_2$ , and  $s_3$  can be reached through jump connections originating from rows  $t_1$ ,  $t_2$ , or  $t_3$  from the previous column. The incoming connection for rows  $s_3$ ,  $s_2$ , and  $s_1$  are indicated by the remaining arrow groups in the illustration.

Equation 3.6, we add connections for possible jumps between each of the blocks in  $\beta$ . To this end, we extend all sets  $Z_{n,m}$  for  $n \in S = \{s_i \mid i \in [1 : I]\}$  by setting

$$\tilde{Z}_{n,m} := Z_{n,m} \cup \{(t_j, m-1) \mid j \in [1 : I]\} \cap Z. \quad (7.3)$$

Furthermore, we set  $\tilde{Z}_{n,m} := Z_{n,m}$  for all other  $n \in [1 : N] \setminus S$ . This adds a new connection from the end position  $t_j$  of each block  $j$  to the start position  $s_i$  of each block  $i$ . We refer to this kind of connection as *jump connection*. Note that no new connection is added from the end of blocks  $i \in [1 : I-1]$  to the beginning of the immediately following target block  $i+1$ , because this connection was already covered by a regular connection. The resulting types of connections in the new sets  $\tilde{Z}_{n,m}$  are illustrated in Figure 7.9. Note, that a set of constraints on the sequence of blocks  $g$  as stated at the end of Section 7.2 can easily be taken into account when creating these jump connections.

As a final step, we perform the modified DTW algorithm using the sets  $\tilde{Z}_{n,m}$  instead of



**Figure 7.10.** Example visualization of a cost matrix and a warping path with jumps obtained via JumpDTW for the same piece of music also shown in Figure 7.4. The piece contains for blocks  $\beta = (\beta_1, \beta_2, \beta_3, \beta_4)$ . The performance block index sequence is  $g = (1, 1, 2, 2, 3, 3, 4, 4, 1, 2)$ .

$Z_{n,m}$ . An example result is depicted in Figure 7.10. From the resulting alignment path with jumps  $p$ , the globally optimum block index sequence  $g$  can be derived easily. Using this JumpDTW approach, we are able to overcome the main weaknesses of the previous two approaches. Giving the alignment path the full flexibility to follow any possible sequence of blocks, the dynamic-programming-based algorithm does the job of delivering a globally optimum sequence  $g$  in a single pass. Neither the disadvantages of the costly iterative greedy strategy of the path degeneration approach that is not guaranteed to find the global optimum, nor the disadvantages of the matching approach that loses robustness by querying each block individually is present in this approach. However, we still need to make some refinements to make it more useful for our real-world scenario.

### 7.5.1 Customizable Start Points and End Points

Since the boundary condition (P1) of the DTW algorithm requires the alignment path to start at  $p_1 = (1, 1)$  and to end at  $p_L = (N, M)$ , currently, the block index sequence  $g$  is implicitly restricted to start with the first block  $\beta_1$  and end with the last block  $\beta_I$ . In practice, however, score tracks can end at a different block, as is for example the case in Figure 7.10. This usually happens in presence of a da capo, where the piece ends at a block marked with the keyword “fine”, which does not have to be the last block in order of notation. To add the necessary flexibility to the JumpDTW algorithm to model this kind of behavior, we modify Equation 3.9, which sets the starting point  $p_L$  for creating the path  $p$  via backtracking and, for convenience, is repeated here as Equation 7.4.

$$p_L = (n_L, m_L) = (N, M) \quad (7.4)$$

Instead of starting the backtracking at the fixed position  $(N, M)$ , we search all block ending positions  $t \in T$  in the  $M$ -th column for the one with the lowest accumulated cost and start

the backtracking from there. More formally,

$$p_L = (n_L, m_L) = \arg \min_{z \in \{(t_i, M) | i \in [1:I]\}} [D(z)] \quad (7.5)$$

$$p_\ell = P_{n_{\ell+1}, m_{\ell+1}}, \quad \ell = L - 1, \dots, 1. \quad (7.6)$$

The algorithm can also be extended to allow start positions that are different from the beginning of the first block in the score sequence. To do so, the following modifications have to be made. When calculating the accumulated cost  $D$ , instead of initializing only the cell  $(1, 1)$  by setting  $D(1, 1) = C(1, 1)$  as in Equation 3.5, we initialize all cells  $(n, 1)$  that can act as starting point by setting  $D(n, 1) = C(n, 1)$  for all  $n \in S$ . Furthermore, we prohibit any incoming connection at these cells  $(n, 1)$  by enforcing  $Z_{n,1}$  to be the empty set. The backtracking of Equation 7.6 may, then, end at any of these cells.

### 7.5.2 Optimum Average Cost

As can be seen from the experiments in [45], when minimizing cost (opposed to maximizing similarity), classic DTW prefers shorter paths with lower total cost but higher average cost over longer paths with higher total cost but lower average cost (the opposite effect happens when maximizing similarity). Even though, in absence of jumps, this preference towards keeping paths short leads to some healthy robustness, it can lead to problems when handling cases with differences in structure using JumpDTW. An example case is illustrated in Figure 7.11(a). The figure shows an alignment path obtained through JumpDTW for a piece where a section in the middle is repeated. The preference of short paths causes the output alignment path to “tunnel” through a region of relatively high cost instead of following the path of lowest average cost. This effect can be eliminated by the following modification of the algorithm. For each cell  $(n, m)$  in the accumulated cost matrix, instead of the minimum accumulated cost  $D(n, m)$  of any path ending at position  $(n, m)$ , we store the accumulated cost of the path with minimum average cost. To do so, for each cell  $(n, m)$ , we also store the length  $L(n, m)$  of the path with minimum average cost ending at position  $(n, m)$ . The algorithm is then performed by first initializing

$$\begin{aligned} D(1, 1) &= C(1, 1) \\ L(1, 1) &= 1 \end{aligned} \quad (7.7)$$

and then iterating

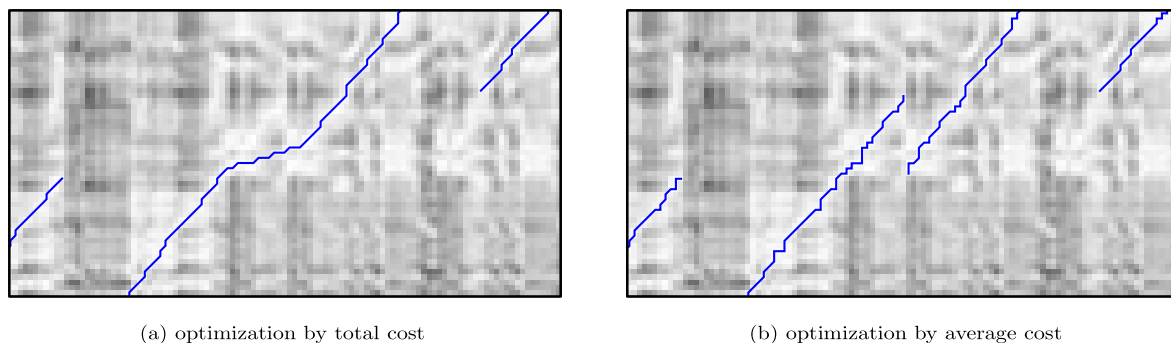
$$P_{n,m} = \arg \min_{z \in Z_{n,m}} \left[ \frac{D(z) + C(n, m)}{L(z) + 1} \right] \quad (7.8)$$

$$D(n, m) = D(P_{n,m}) + C(n, m) \quad (7.9)$$

$$L(n, m) = L(P_{n,m}) + 1. \quad (7.10)$$

over  $n, m$ . Note that in Equation 7.8 we minimize over the average cost of any path ending at position  $(n, m)$ , which we also refer to as *step decision cost*. In Equation 7.9, however, we save the accumulated cost instead of the step decision cost. Using this modification, the algorithm outputs an alignment path with optimum average cost and no longer prefers shorter paths with less accumulated but higher average cost. The result of the modified algorithm in the example case is depicted in Figure 7.11(b).



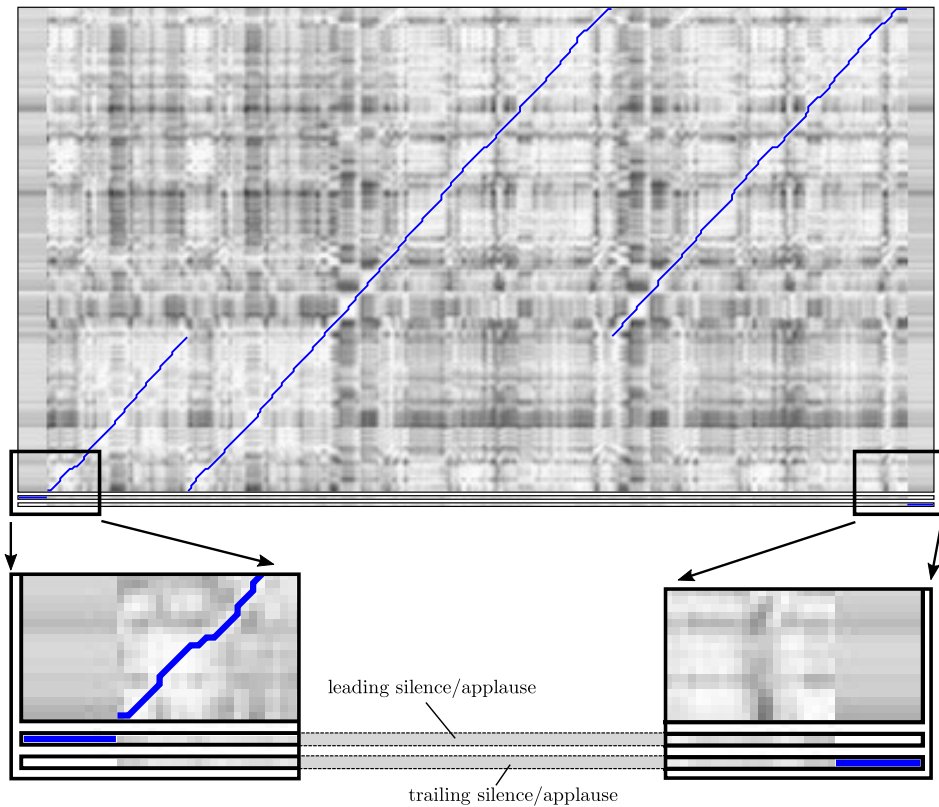


**Figure 7.11.** Example case where the optimization by average cost outperforms the optimization by total cost. Subfigures (a) and (b) show the same excerpt of a local cost matrix. Subfigure (a) additionally shows the corresponding excerpts of a warping path with jumps computed using the regular optimization by total cost. Subfigure (b) shows the corresponding excerpts of a warping path with jumps computed using the optimization by average cost. In case (a), a repeat is missed out because the sum of the cost along the erroneous section of the path is lower than the sum of the cost along the (longer) correct section. The path in case (b) correctly captures the repeat. Since there is no longer a preference towards keeping the path length short, the path in (b) appears less smooth because it contains more horizontal and vertical steps instead of diagonal steps.

### 7.5.3 Special States

The HMM way of modeling the realization of the score track as an audio recording allows for adding special states to account for sections in the audio that are not represented in the score. Such special states can use custom cost functions that should deliver low cost values when comparing the special state to audio feature frames that are to be modeled by the state, and high cost values otherwise. An example usage for such special states would be to model cadenzas that are not written in the score. Throughout the cadenza in the audio recording, the HMM could reside in that special state until the performance of the regular score is continued.

To incorporate special states for the score track into our JumpDTW approach, we extend the cost matrices by additional rows at the bottom, with each row representing one special state, see Figure 7.12. The cells of these rows representing the special states are incorporated in the dynamic programming algorithm by adding special connections from or to regular cells. In our implementation, we use two special states. The first is labeled “leading silence/applause” and models silence or applause at the beginning of the audio recording before the content of the score is performed. The second special state is labeled “trailing silence/applause” and models silence or applause at the end of the recording after the performance of the score is finished. Both states use the same local cost measure as the regular states to compare each feature frame from the audio sequence to a normalized chroma vector with equal distribution, which models both silence and applause. The cells of the row resulting from the special state “leading silence/applause” are connected to the cells of the first row of the regular cost matrix so that after starting in this special state, the alignment path is free to jump over to the regular score sequence at any position in the audio sequence. Once the state is left, the path can never return to this state. The cells of the row for the special state “trailing

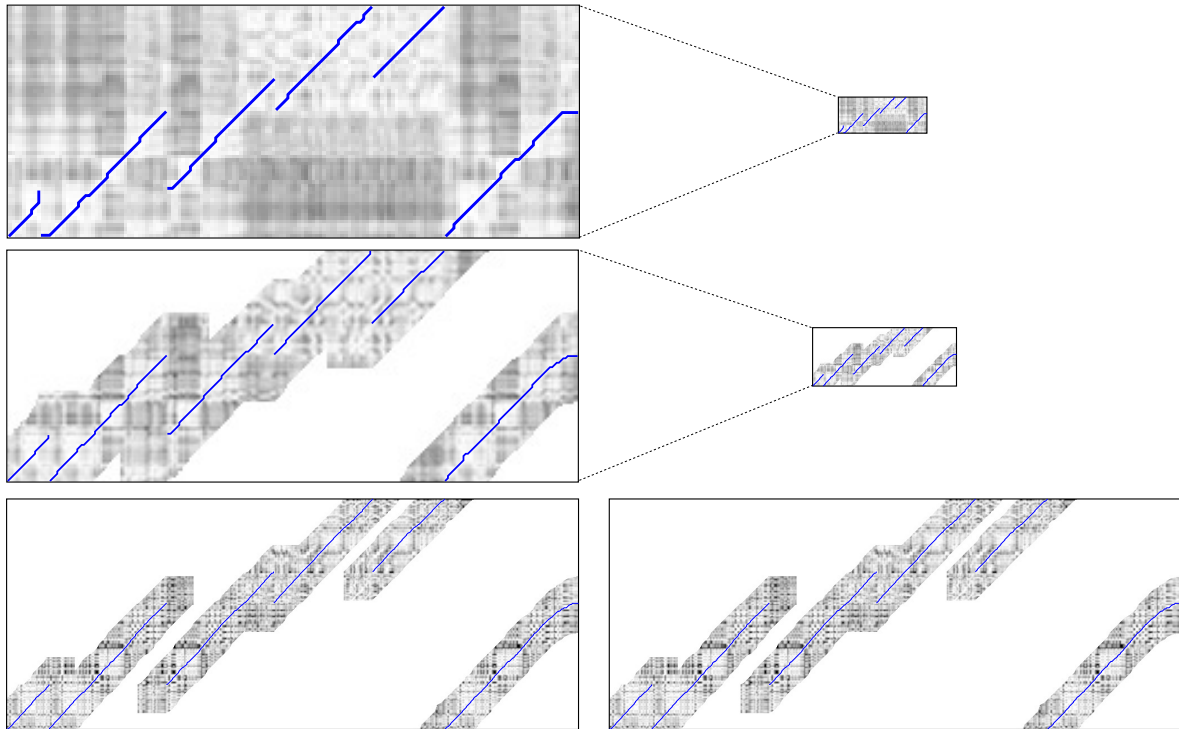


**Figure 7.12.** JumpDTW result for a case where the audio recording comprises rather long silence before the beginning and after the end of the performance. Two single-row cost matrices below the main cost matrix represent the two special states for leading and trailing silence/applause in the visualization. As can be seen in the magnifications of the bottom left and bottom right corners, the alignment path starts in the special state for leading silence/applause and jumps into the main matrix as soon as the performance starts. After the performance ends, the path jumps into the special state for trailing silence/applause.

silence/applause” get an incoming connection from each of the end positions of the score blocks where the score track is allowed to end. Once the path enters this special state, it can never go back to any of the regular states. Within the rows for the special states, the cells are simply left-to-right connected so that the alignment path can stay inside that state for any amount of audio frames. A visualization of a local cost matrix including the two special states at the bottom is depicted in Figure 7.12.

#### 7.5.4 Multi-Scale Approach

To improve the efficiency of the JumpDTW algorithm, we can make use of the same kind of multiscale approach as has been used in [100] to significantly increase the efficiency of classic DTW. Here, the idea is basically to perform the algorithm in multiple passes with increasing feature resolution. In the first pass, the full cost matrix is calculated, but since the feature resolution is relatively low, the computational cost is not very high. In each subsequent



**Figure 7.13.** Illustration of the multiscale JumpDTW approach. The left column shows the cost matrix and the alignment path with jumps for three resolutions. For the coarsest resolution, which is depicted at the top, the complete cost matrix is calculated. In the higher resolutions, however, the cost matrix is calculated only for neighborhood around the path obtained from the previous resolution. The right column shows the same matrices and paths as the left column but with keeping the sizes of the visualizations proportional to the sizes of the matrices. From this visual impression one can get an intuitive understanding to what degree this multiscale approach can reduce the computational complexity and memory requirements.

pass, the alignment path that is obtained from the previous pass is used to constrain the calculations to be performed only in a neighborhood around that path. This means that for the passes with higher resolution the cost matrix on which the algorithm is performed is sparse. Since, unlike in classical DTW, in JumpDTW alignment paths are allowed to incorporate jumps, the data representation of the sparse matrices must be flexible enough to handle those. In our implementation, we represent the sparse matrices as a set of tubes that follow the alignment path of the previous pass. An example illustration is depicted in Figure 7.13. Besides a significant speedup of the computations, due to the lower memory requirements, this multiscale approach enables the use of JumpDTW even for longer tracks (e.g. 30+ minutes) at a target resolution of 10 Hertz on a machine running a 32-bit Java virtual machine.

	mch blk % (#)	ins blk % (#)	omt blk % (#)	mch bar % (#)	ins bar % (#)	omt bar % (#)	prf % (#)
nojumps	70.2 (214)	0.3 (1)	29.8 (91)	74.6 (8831)	0.0 (1)	25.4 (3005)	69.8 (8258)
s1_plain	93.4 (285)	9.2 (28)	6.6 (20)	99.2 (11740)	0.9 (105)	0.8 (96)	98.5 (11661)
s2_add_special_states	93.4 (285)	5.6 (17)	6.6 (20)	99.3 (11759)	0.7 (82)	0.7 (77)	98.8 (11692)
s3_penalize_0.5_100	94.4 (288)	9.5 (29)	5.6 (17)	99.4 (11767)	0.4 (51)	0.6 (69)	99.1 (11725)

**Table 7.1.** Evaluation results for classical DTW and different variants of JumpDTW.

## 7.5.5 Experiments and Results

To evaluate the usefulness of JumpDTW in a practically relevant application, experiments are conducted on the first 15 piano sonatas by Beethoven including a total of 54 individual movements. The score data is obtained from OMR results of a printed sheet music edition, and the performances are given as audio CD recordings. Since the score data does not include any tempo information, a mean tempo is estimated for each movement using the number of bars and the duration of the corresponding performance. For each movement, the notation bar sequence  $\nu$  is known and the score block sequence  $\beta$  is obtained using block boundary indicators extracted from the score. Note that this may include block boundary indicators where actually no jump or repeat occur in the performance. The performance bar sequence  $\pi$  is given as ground truth and is used to derive a ground truth block index sequence  $g$  with respect to  $\beta$ . For our test data set, the total number of score blocks appearing in the sequences  $\beta$  of the 54 movements is 242. The total number of notation bars is 8832. Note that, because of repeats and jumps, a score block may occur more than once in the performance. Therefore, the total number of blocks appearing in the sequences  $g$  is 305 which corresponds to a total of 11836 bars being played in the performance. The total duration of the performance amounts to 312 minutes.

JumpDTW is performed on the data using  $\beta$  as described in Section 7.5. From the resulting alignment path with jumps, an output block index sequence  $g' = (g'_1, \dots, g'_{j'})$  is obtained. In the optimal case, this block index sequence  $g'$  would be equal to the ground truth block index sequence  $g$ . Table 7.1 shows the results of comparing  $g'$  to  $g$  using several different evaluation measures. Each row shows the results for different sets of  $g'$  obtained using a different JumpDTW variant. Each entry in the table summarizes the results for all 54 movements. The first row, tagged `nojumps`, represents the results when using classical DTW as described in Section 3.2 to align the plain block sequence  $\beta = (\beta_1, \dots, \beta_I)$  without any jumps or repeats to the performance. This corresponds to an output block index sequence  $g' = (1, \dots, I)$ . The results of the comparison between this  $g'$  and the ground truth  $g$  serve as bottom line in our evaluation. The second row, tagged `s1_plain`, represents the basic JumpDTW algorithm as described in Section 7.5 including the relaxed boundary condition for da capo/fine cases as described in 7.5.1.

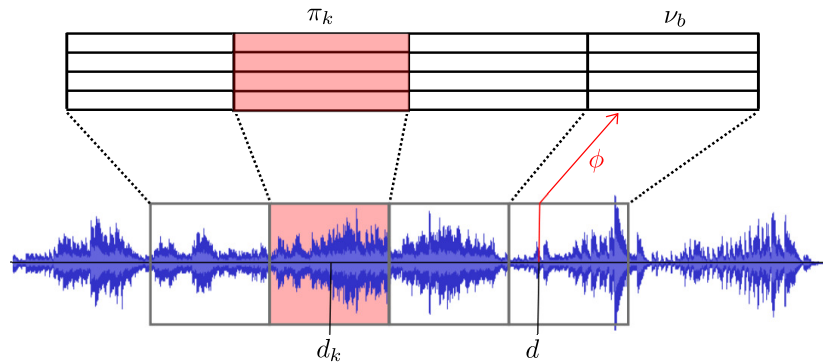
The numbers plotted in the first six columns are based on a direct comparison of the sequences  $g'$  and  $g$  and measure how many blocks (abbreviated as `blk`) or performance bars (`bar`) match between the two sequences (`mch`), have been erroneously inserted into  $g'$  (`ins`), or have been erroneously omitted in  $g'$  (`omt`) with respect to the ground truth  $g$ . To this end, we calculate an optimum alignment between the two block index sequences using a variant of the edit distance that only allows insertions and deletions (but not replacements). To find an alignment between the two block index sequences that is optimal with respect to the amount

of inserted and omitted bars (instead of blocks), each block index entry in the sequences is weighted by the length of the corresponding score block. Each entry in Table 7.1 is given as a percentage with respect to the total number of blocks/bars in the performance followed by the absolute number in parentheses. For example, the entry 70.2(214) in the first row and column means that 214 blocks of  $g'$  have a matching counterpart in  $g$ , which is  $214/305 = 70.2\%$  of the total number of blocks in  $g$ . Similarly, the entry 74.6(8831) for matching bars means that the 214 matching blocks have a total length of 8831 bars, which is  $8831/11836 = 74.6\%$  of the total length of  $g$  in bars.

A further evaluation measure (**prf**), which is plotted in the last column of Table 7.1, expresses the alignment accuracy on the bar-level. This measure is motivated by the application of visually presenting sheet music that is linked on a bar-wise level to a given recorded audio performance. For this application, we want to measure for how many of the performance bars the alignment computed via JumpDTW is suitably accurate. To this end, the ground truth block index sequence  $g$  is used to create a feature sequence  $x$  from the score data that matches the repeats and jumps of the performance. Then, this feature sequence is synchronized to a feature sequence  $y$  obtained from the performance using classical DTW. From the output alignment path, we derive a bar-wise score-performance synchronization that maps each performance bar  $\pi_k \in \pi$  to a temporal region with center time  $d_k$  in the performance, see Figure 7.14. Furthermore, this synchronization delivers a mapping  $\phi : [0, D] \rightarrow [1 : B]$ , with  $D$  being the duration of the performance, that for each time positions  $d \in [0, D]$  in the performance returns an index  $b \in [1 : B]$  indicating that bar  $\nu_b$  is played at time  $d$ , see also Figure 7.14. Since, from manual inspection, the synchronization results obtained when using the ground truth block index sequence  $g$  are known to be suitably accurate on a bar-wise level, they are used as a reference for finding deviations in the synchronization results obtained using  $g'$ . For each performance bar  $\pi_k$ , we take the bar center time  $d_k$  and input it into the mapping  $\phi'$  obtained from the synchronization results using  $g'$ . The performance bar is counted as correctly matched if  $\phi'(d_k) = \phi(d_k)$ , which means that in the synchronization obtained using  $g'$ , the time position  $d_k$  points to the same bar  $\nu_b$  as in the reference synchronization. Unlike the mere number of matched bars listed in the column **mch bar**, this measure takes into account the extra confusion that is caused in the synchronization by erroneously inserted or omitted bars.

From the results using classical DTW (**nojumps**) one can see that about 70–75% of the blocks and bars of the performance are covered by the plain score bar sequence. The remaining 25–30% are repeats that are omitted in this sequence. One single block with a length of 1 bar is counted as inserted because of a repeat with alternative endings that is not played in the performance. The synchronization-based measure indicates a similar result: 69.8% of the center time positions of the bars in the performance were aligned to the correct bar in the score. These results are improved significantly, when using JumpDTW (**s1\_plain**). Here, 93.4% of the blocks and 99.2% of the bars are matched correctly. In the synchronization-based measure, 98.5% of the performed bars match the reference synchronization. Even though 28 blocks have been erroneously inserted and 20 blocks have been omitted, this amounts to only 105 inserted bars and 96 omitted bars, revealing that the mean length of inserted and omitted blocks is only about 4.2 bars.

Manual inspection of the results for the individual movements reveals that in many cases



**Figure 7.14.** Illustration of a bar-wise score-performance synchronization. Each performance bar  $\pi_k$  is synchronized to a temporal region of a performance with bar center time  $d_k$ . Furthermore, a mapping  $\phi$  can be derived that for a given time position  $d$  in the performance outputs the index  $b$  of the corresponding bar  $\nu_b$  in the notation bar sequence.

an extra block is inserted at the beginning or the end of the sequence to cover for silence at the beginning or end of the performance. In one case, this even leads to the last block of the sequence being confused with an incorrect one. To encounter this issue, we extend the JumpDTW algorithm by adding special states to the score representation that model silence at the beginning or end of the performance as described in Section 7.5.3. The results for this modification are listed in the line labeled `s2_add_special_states` and show slightly improved numbers. An in-depth analysis of the results shows that this modification solved all of the previously mentioned problems caused by initial or trailing silence in the performance. Furthermore, it turned out that 130 of the  $82 + 77 = 159$  inserted and omitted bars occur in just 3 of the 54 movements. The performance of the first movement of “Sonata 8, Op. 13, *Pathétique*” contains extreme tempo changes with slow sections of roughly 20 BPM (beats per minute) alternating with fast sections of about 300 BPM. This results in a large difference between the estimated mean tempo of the score and the tempo of the slow sections in the performance. The JumpDTW algorithm reacts by erroneously inserting more or less random blocks to cover the unexpectedly slow sections of the performance. A different kind of problem occurs in “Sonata 12, Op. 26, *Andante con variazioni*”. Here, the second block is a variation of the first block that has virtually the same harmonic progression. The JumpDTW erroneously treats this second block in the performance as a repeat of the first block in the score. This behavior is not very surprising considering that the content-based comparison of score and performance is somewhat noisy and for the chroma-based features used, sections with the same harmonic progression are almost indistinguishable. In “Sonata 13, Op. 27 No. 1, *Andante-Allegro*” it is again a significant change in the tempo that causes a problem. Here, a repeat of a block (length = 9 bars) of the faster Allegro section is omitted by JumpDTW, which is provoked by the estimated tempo of the score being significantly slower than the tempo of the corresponding section of the performance. For all of the remaining movements, only blocks of length 2 or lower are inserted or omitted.

To encounter the problems discussed above, we further extend the JumpDTW approach by introducing a penalty cost for performing jumps in the alignment path that is added to the accumulated cost. The cost value is set to  $0.5 \cdot \frac{N}{100}$ , with  $N$  being the length of the score feature sequence. The particular formula is motivated by the idea of choosing a cost value

that is close to the cost of matching 1/100-th of the score to a section of the performance that is not considered similar. Since in our implementation, we use normalized chroma features with a cosine measure for the local cost, a local cost value of 0.5 is already considered not similar. The results for this modification are listed in the row `s3_penalize_0.5_100`. A closer analysis shows that adding the penalty solves the confusion for the “Andante con Variazioni” and lowers the amount of inserted bars for the slow sections of the “*Pathetique*”, which leads to a better overall result. However, the penalty also causes degradation for many of the other movements because short blocks for alternative endings are no longer skipped. Tuning the penalty cost to higher or lower values did not improve the situation. An increased penalty led to an increased amount of erroneously skipped short blocks while a decreased penalty no longer solved the confusion for the two movements discussed above.

### 7.5.6 Summary of the JumpDTW Approach

The JumpDTW approach for computing the partial synchronization for given score tracks and audio tracks in our scenario combines several favorable properties. Firstly, it is guaranteed to output a globally optimum block index sequence  $g$ . What optimality criterion is used can be adjusted by the user. Using a multiscale implementation, the algorithm performs very efficiently in terms of computational complexity and memory requirements. Special states can be used to model sections of the audio recording that are not present in the score. This has been shown to work well for silence at the beginning and end of the recordings. A similar method could be used to model cadenzas with known position but unknown content in the score. Since the algorithm finds an optimum sequence in presence of any amount of candidate jumps, an approach to deal with the missing information on repeats with alternative endings in the score data would be to simply guess forward jump connections for each of up to a fixed number of bars before double bar lines with closing repeat signs. Then, the algorithm would choose to find the jumps that best suit the given audio recording. If the score data is accurate and the measure for comparison works well, the algorithm is expected to take the correct jumps. However, in practice, noise and errors in the data as well as the relatively coarse measure for comparison we used in our scenario cause the approach to become less robust with increasing amounts of candidate jumps.

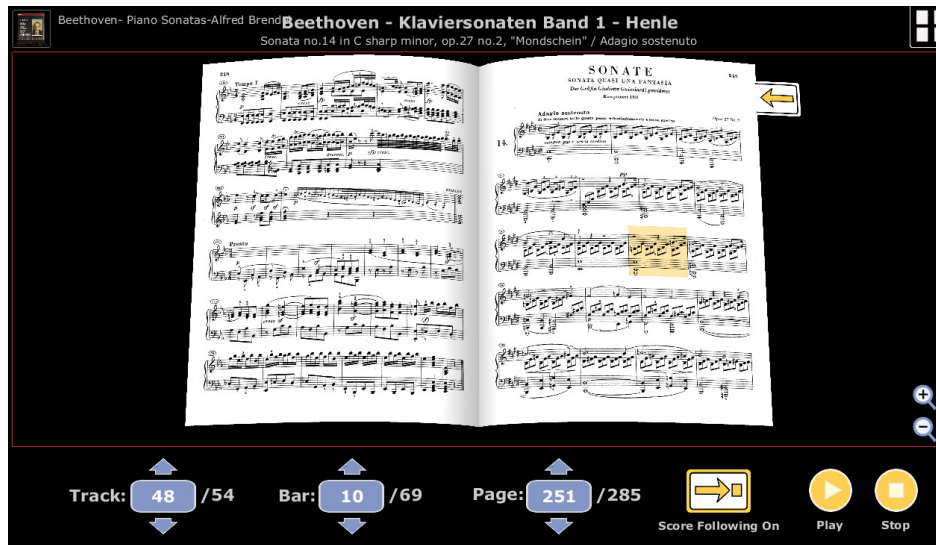




## Chapter 8

# Applications

Using the methods presented in the previous chapters of this thesis, we are able to calculate temporal alignments for given scanned sheet music documents and corresponding audio recordings. Furthermore, we are able to identify all documents that correspond to the same piece of music. Since the proposed methods are mostly automatic and require only little manual interaction, alignments and links between the documents can be obtained for large collections of music documents. In this chapter, we present applications and user interfaces for presenting the music documents and explicitly making use of the available temporal alignments and links. A main concept in the presentation of the documents to the user is that only the high quality input documents are used for presentation. All the intermediate data, especially the error-prone symbolic score data obtained via OMR, are never directly visible to the end-user. The target use case of the presented applications is that of a digital music library where users can access the digitized collection of sheet music scans and audio recordings. This scenario is being pursued as part of the PROBADO project, see also Section 4.1. A prototypical user interface has been designed and implemented. This user interface and the functionalities it offers are presented in the individual sections of this chapter. Beyond that, an introductory demo video of the interface can be found on the website of the author<sup>1</sup>. Section 8.1 introduces the *score viewer* interface for displaying and interacting with the scanned sheet music documents. In Section 8.2, we present the *audio viewer* interface for visualizing and interacting audio CD collections. Both interfaces can then be used for presentation of and navigation in the content across the domains of sheet music and audio, which is described in Section 8.3. If more than one audio recording is available for a single piece of music, the user can switch between the available interpretations as described in Section 8.4. Finally, the interface can be used to perform content-based retrieval across the domains, which is presented in Section 8.5.



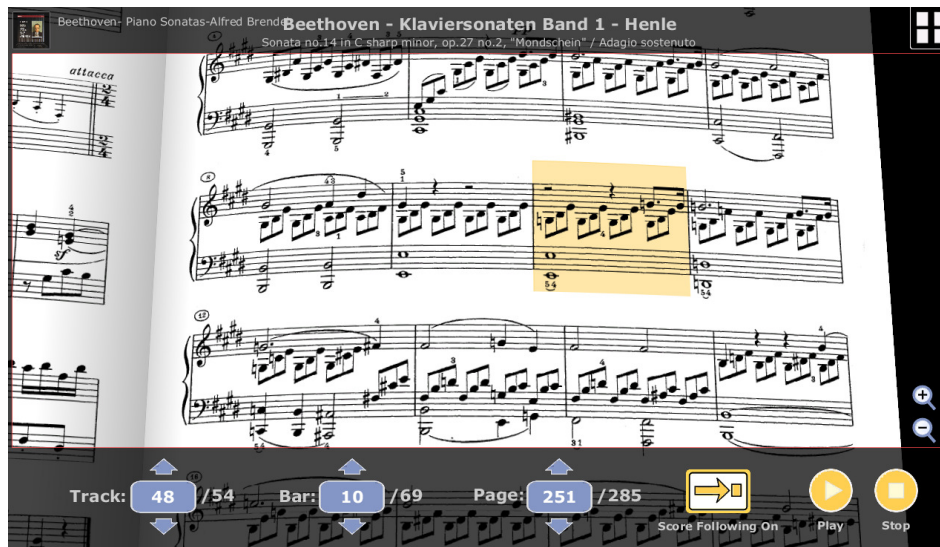
**Figure 8.1.** Screenshot of the score viewer showing a the sheet music book published by G. Henle that contains the first 15 piano sonatas by Beethoven.

## 8.1 Score Viewer

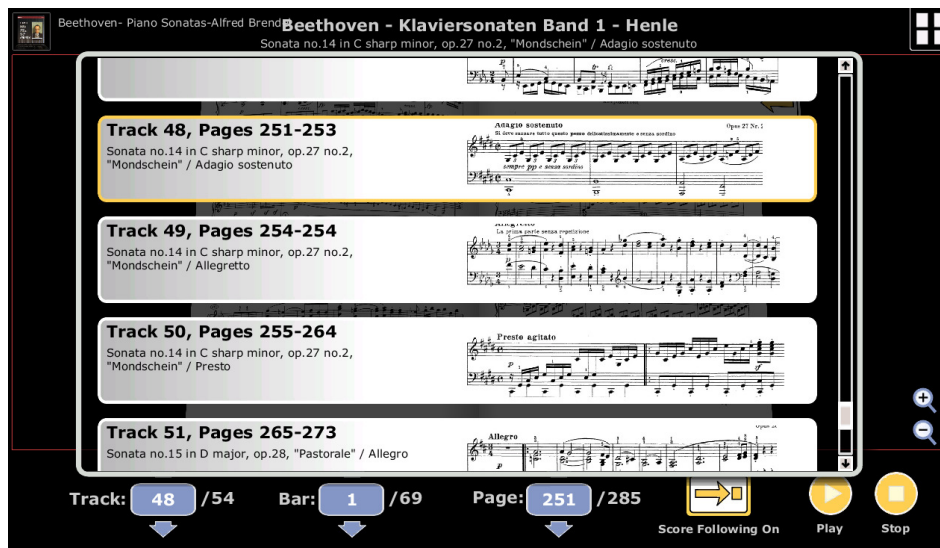
The *score viewer* displays sheet music in form of a 3D virtual book, see Figure 8.1. The advantage of this style of display is that it resembles the original look and feel of the sheet music book that is represented by the given individual scans for each page. This makes interacting with the content more natural and intuitive for the user. The view on the book can be zoomed and scrolled freely and the tilt angle can be adjusted to suit the viewers preference. The view settings selected in Figure 8.2 show a more close-up view of the right page shown in Figure 8.1. A translucent yellow box is used to indicate the current playback position in score book on a bar-wise level. The features related to playback are discussed in more detail in Section 8.3. The user can navigate freely throughout the whole sheet music book. To this end, pages can be turned by clicking the left or right borders of the book using the mouse pointer. A bookmark at the side of the book (upper right side in Figure 8.1) always points to the current playback position. If the user has navigated away from the current playback position, clicking this bookmark causes the viewer to instantly go back to the page of the current playback position.

At the top of the score viewer, a title and other meta information about the current score track are displayed. Control fields for the track number, page number, and the performance bar number in the track are shown at the bottom. In addition to displaying the current playback position in the book, these controls can be used to navigate to the previous or next track, page, or bar, or to enter a specific track number, page number, or bar number using the computer keyboard. A click on the icon at the top right opens the so-called *track start browser*, which is depicted in Figure 8.3. Similar to the table of contents shown in Figure 6.7, the track start browser shows a list of all score tracks represented by their title and their

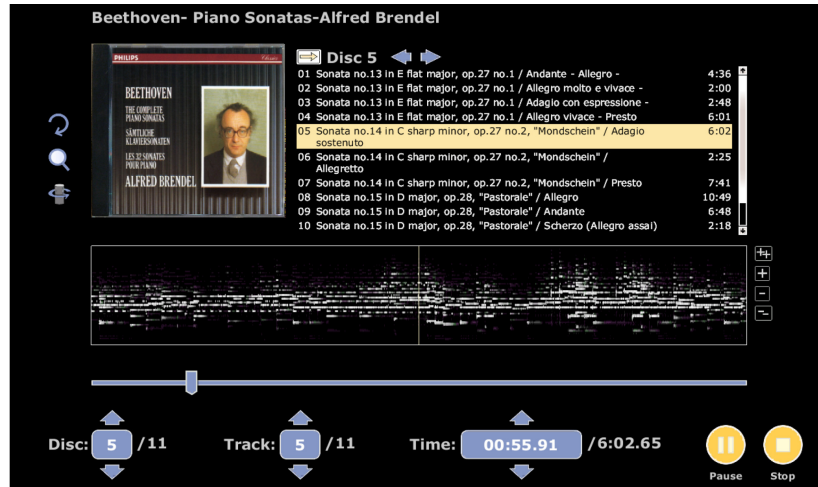
<sup>1</sup><http://www.iai.uni-bonn.de/~fremerey/>



**Figure 8.2.** The view on the sheet music book displayed in the score viewer can be zoomed and scrolled freely.



**Figure 8.3.** The track start browser allows quick access to all the score tracks contained in the sheet music book.



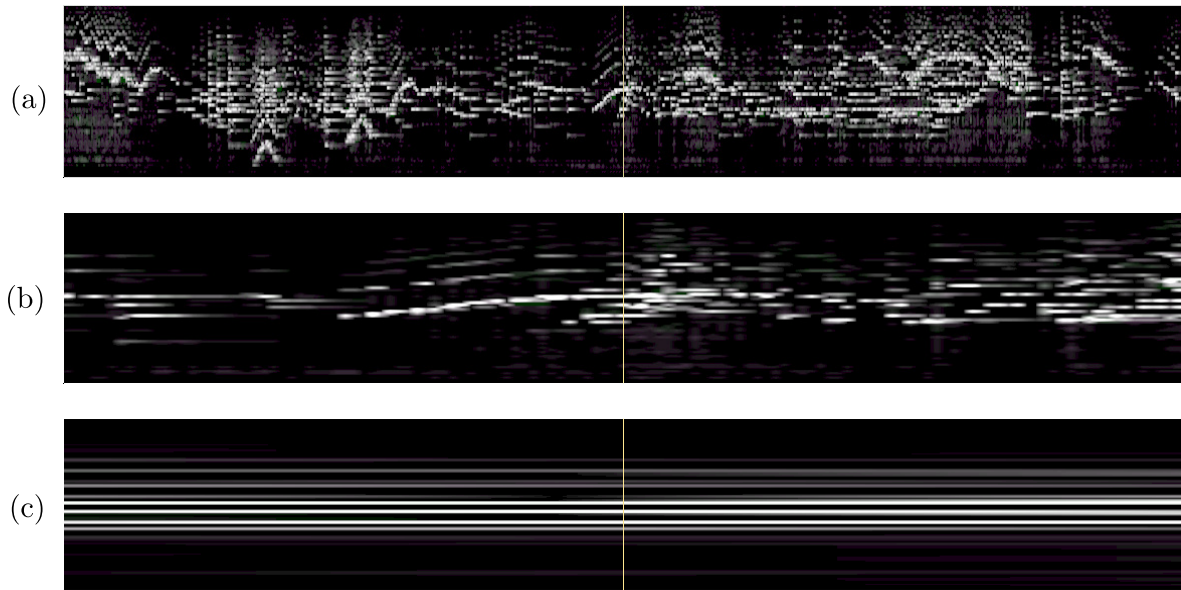
**Figure 8.4.** Screenshot of the audio viewer showing an audio CD collection of the complete piano sonatas by Beethoven performed by Alfred Brendel.

first grand staff in the score. Note that using the spatial information obtained from the OMR results, the image data for these excerpts are extracted from the scans automatically. Using the track start browser, the user can quickly navigate to any of the score tracks contained in the sheet music book. Additional controls for playback and interpretation are found in the top left and bottom right corners. These will be explained in the subsequent sections of this chapter.

## 8.2 Audio Viewer

The *audio viewer* employs a similar design as the score viewer to allow the interaction with audio CD collections, see Figure 8.4. A virtual CD case is displayed at the upper left part to give the user the feeling of interacting with a real physical medium. The case can be magnified and turned around to display the backside of the cover. A list of CD tracks showing a title and duration is displayed on the right side together with a selector for the disk number. Using these controls, the user can browse through all disks and tracks that are part of the collection.

The acoustic content of the current audio track is visualized in the center part of the audio viewer. The visualization shows a time-frequency representation of the acoustic content. Time is represented on the horizontal axis and frequency is represented on the vertical axis. The frequency axis uses a logarithmic scale, which means that octaves are equally spaced. A vertical line with fixed position at the center of the visualization area indicates the current playback position. Accordingly, during playback, the visualized content moves from right to left. A spot with bright color in the visualization means that the corresponding frequency is present with high amplitude at the corresponding time. Areas with dark color indicate that the corresponding frequencies are present with only low amplitude or not present at all. This visualization is computed using a short-time Fourier transform in combination with binning and thresholding strategies. The result looks somewhat similar to a piano-roll display, which



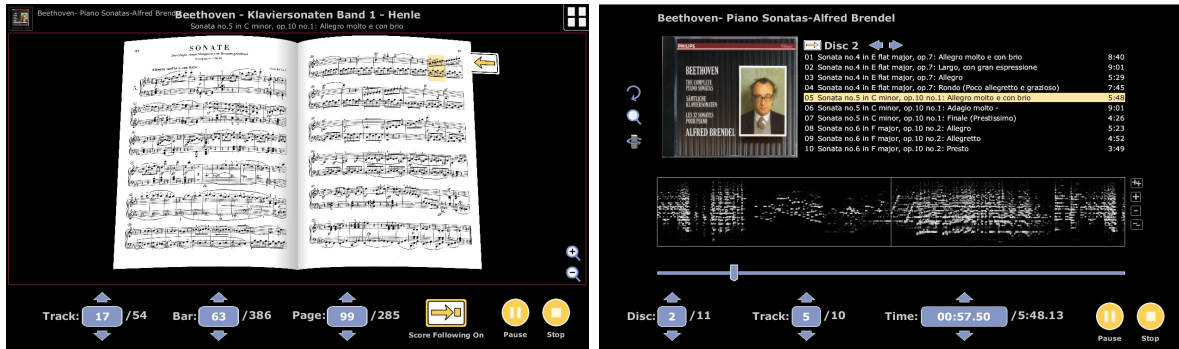
**Figure 8.5.** Illustration of three different zoom levels for the visualization of audio content used in the audio viewer. A vertical line at the center of the visualization marks the current playback position. (a) The regular zoom level gives an overview of about one minute of audio content. (b) Same playback position as in (a) but zoomed in, so that only a few seconds of audio content are displayed. (c) Same playback position as in (a), but with maximum zoom setting. The horizontal lines seen in the visualization can be interpreted as a picture of the instantaneous frequency content.

is often used to visualize MIDI data, see for example [86, 76]. However, in contrast to a piano-roll display, a single long note that is played in the performance is usually displayed as multiple horizontal lines, because in this rather simple implementation harmonics of the fundamental frequency are not suppressed as in more sophisticated approaches like [127]. The time axis of the visualization can be scaled to different zoom levels by the user. In the regular setting shown in Figure 8.5(a), the visualization gives an overview of about one minute of the acoustic content surrounding the current playback position. In the maximum zoom setting, which can be accessed by clicking the double plus symbol in the viewer, the visualization corresponds to the instantaneous frequency content at the playback position, see Figure 8.5(c).

Controls at the bottom section of the audio viewer offer a similar functionality than those used in the score viewer. Here, instead of score track, page, and bar number, the controls allow access to the audio track, disk number, and track number on the disk. In both the score viewer and the audio viewer, controls at the bottom right can be used to start and stop the playback.

### 8.3 Cross-Domain Presentation and Navigation

A key concept of the presentation of the data to the user is that the score viewer and audio viewer are linked together through the sheet music-audio synchronization. Instead of acting

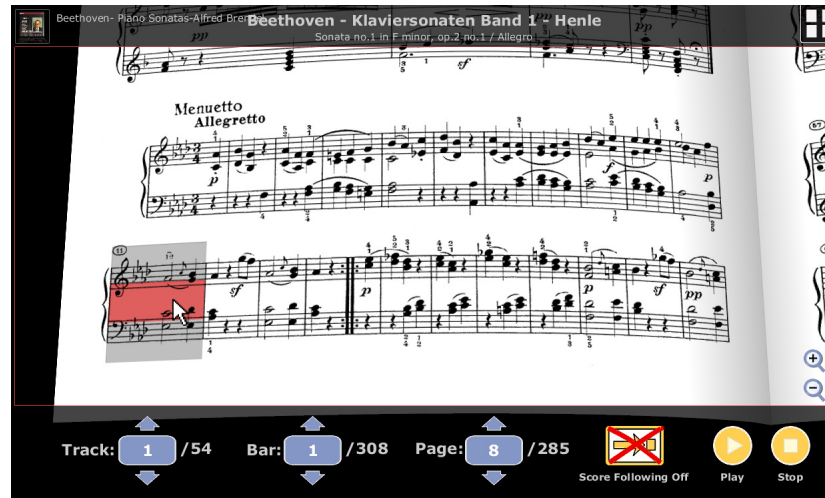


**Figure 8.6.** Illustration of simultaneous presentation of musical content in the score viewer and audio viewer. The score viewer (left) shows the current playback position as a yellow overlay box over the current bar in the sheet music book (near the top right corner of the book). The audio viewer (right) shows the corresponding position in the visualization of the acoustic content and on a slider representing the timeline of the complete track. While playback time advances, the content in the visualization moves towards the left, and the highlighted bar in the sheet music book progresses throughout the book. Pages in the book are turned automatically.

as just a viewer for sheet music and a player for audio recordings separately, the linked combination of the two appears more like two different windows of presentation of and interaction with the same piece of music. Since a single piece of music is presented and can be accessed in two different domains, i.e., the sheet music domain and the audio domain, we use the attribute *cross-domain* to describe the applications that emerge from the combination of the two.

A typical first application is cross-domain presentation, which, in our case, means letting the user follow the score while simultaneously listening to the audio playback. Since the software is aware of the current playback position in the sheet music, it can assist the user in following the performance by continuously updating the highlighted bar in the sheet music book according to the playback position, see Figure 8.6. The score viewer offers a special mode tagged “score following”. If this mode is active, the viewer automatically controls page turning and scrolling to keep the currently played bar visible in the sheet music book. The score following mode is turned off automatically if the user navigates away from the current playback position by manually turning a page.

A second application that makes use of the sheet music-audio synchronization is cross-domain navigation. Since the sheet music domain and the audio domain are linked, changing the playback position in one domain also affect the playback position in the other domain. A very useful application of this is to navigate to a particular position in the audio recording by selecting the corresponding bar in the sheet music. The compact visual representation of the music that is given by the sheet music makes finding a particular position of interest in a piece of music much easier and faster than searching for the corresponding time position in the audio recording. Using the score viewer, the user can navigate to any position in the piece of music by clicking the target bar with the mouse pointer. The audio viewer will, then, update the playback position in the audio recording accordingly. Note that a bar in the notation bar sequence of the sheet music can occur more than once in the performance. If



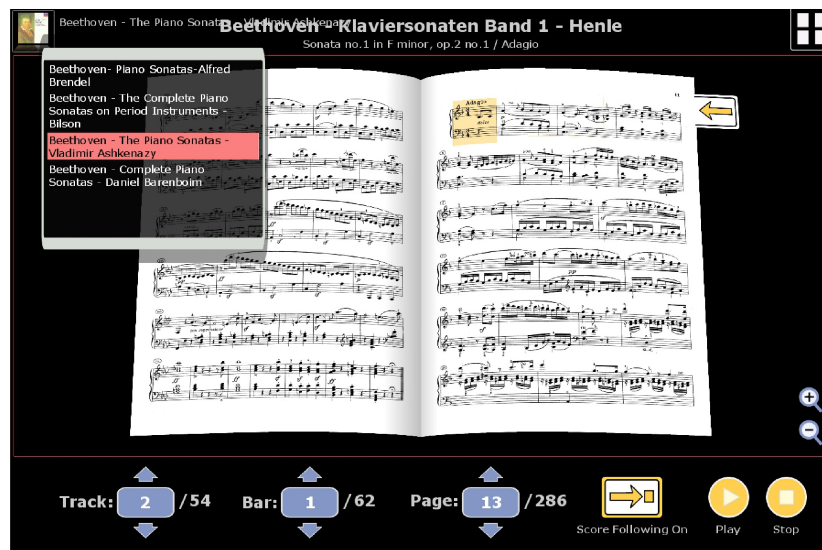
**Figure 8.7.** The highlighting box of the bar under the mouse cursor is split into three horizontal slices indicating that this bar occurs three times in the performance. By clicking on the second slice from the top, the user directs the player to jump to the second occurrence of the bar in the performance.

this is the case, the score viewer indicates this by splitting the highlighting box of the target bar that appears when the mouse cursor hovers the bar into horizontal slices, see Figure 8.7. Each horizontal slice represents one occurrence of the bar in the performance in order from top to bottom. By clicking on the second slice from the top, for example, the user directs the user interface to jump to the second occurrence of the bar in the performance.

The score viewer and audio viewer are not only linked on the level of bars as provided by the sheet music-audio synchronization, but also on the level of leaf-level works. If the user navigates to a different score track in the score viewer, a corresponding audio recording is loaded in the audio viewer if available. If more than one corresponding audio recording is available, one of them is chosen as a default. The user also has the option to navigate to a different audio track using the audio viewer. In that case, the score viewer jumps to the beginning of the corresponding score track in the currently loaded sheet music book if the selected target work is contained. Otherwise, if available, a different sheet music book containing that work is loaded and displayed.

## 8.4 Switching Interpretations

In many cases, more than one interpretation or recording of a given piece of music is available. If all of these interpretations are synchronized to the same reference sheet music representation, the synchronizations can be used to enable a mechanism for convenient switching of interpretations during playback without changing the playback position. To this end, the score viewer offers access to an interpretation selector at the upper left corner, see Figure 8.8. The user can change the interpretation by selecting one of the available options from the list. The player, then, remembers the current bar and occurrence number, loads the target



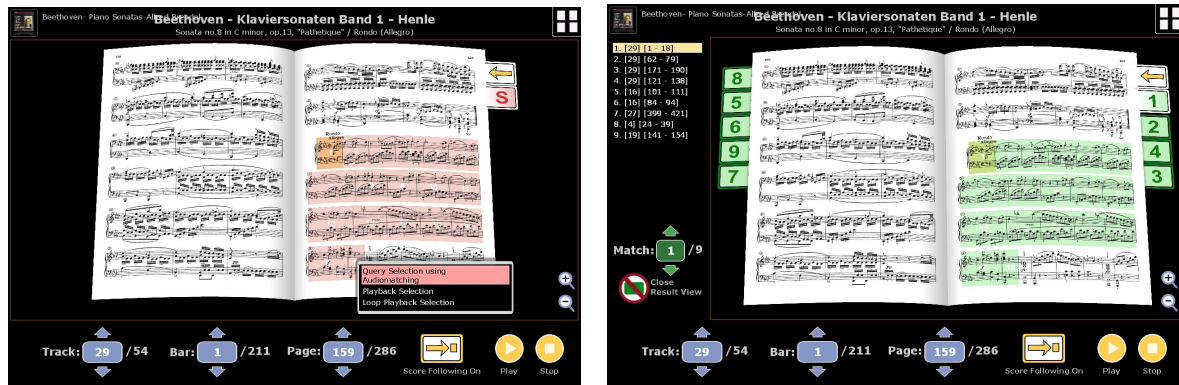
**Figure 8.8.** If more than one audio interpretation of the current score track is available, the interpretation can be switched using a list that is opened by a click on the icon at the top left of the score viewer.

interpretation, and starts the playback at the time position that corresponds to the remembered bar and occurrence number in the new interpretation. Using this mechanism, the same sections of different interpretations can be compared conveniently without having to face the distraction of having to search for the right time position in the different recordings.

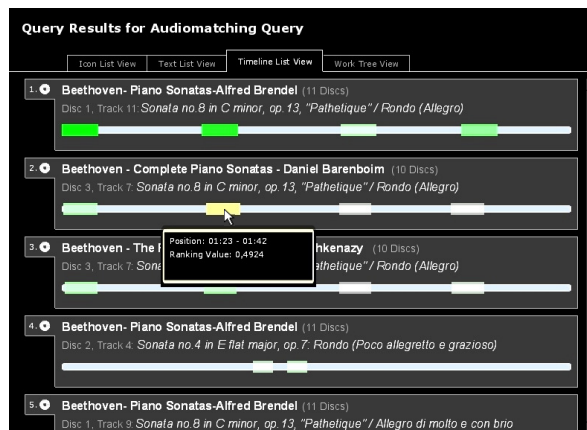
## 8.5 Cross-Domain Retrieval

Besides for presentation and navigation, the user interface presented in this chapter can also be used for content-based retrieval. To this end, the user can choose a section of the performance as a query by selecting a sequence of performance bars from the sheet music. Bars in the sheet music are selected in the same way as text in a standard text editor. The user clicks-and-holds on the first bar of the intended section and drags the pointer to the last bar of the intended selection. The selected bars are highlighted in translucent red color with the indices of the bars in the performance bar sequence being indicated at the center of each bar, see the left part of Figure 8.9. A right-click on the selected bars opens a context menu, from which the user selects the option “Query Selection using Audio Matching”. This triggers a content-based search using the audio data of the performance corresponding to the selected bars as a query. In the example shown in Figure 8.9, the query consists of the first theme of the Rondo of Beethoven’s Piano Sonata No. 8, Op. 13, “Pathétique”, which appears four times throughout the piece. The database that is used to search in is an index created from the audio data of all audio CD collections that are available through the system. For the content-based retrieval, a variant of a search technique called *audio matching*, which in its approach and output is somewhat similar to using subsequence dynamic time-warping as described in Section 3.5, is used. Details on audio matching techniques can be found in [75].





**Figure 8.9.** Left: A sequence of performance bars is selected in the sheet music book to formulate a content-based query. Right: The query results are displayed as regions highlighted in green color in the book and as bookmarks at the side of the book.



**Figure 8.10.** The query results of the content-based retrieval are displayed as a list of timelines with matches indicated as green boxes.

As a result, a set of matches is obtained, with each match specifying an audio track, a temporal region in that audio track, and a ranking value that expresses how similar the specified region is to the query. The matches that are found in the currently loaded audio CD collection are displayed as corresponding highlighted regions in the sheet music book, see the right side of Figure 8.9. Additionally, each match gets its own bookmark at the side of the book and is displayed as an entry in a result list on the left side of the score viewer. The complete set of matches, i.e., the matches that were found in any of the available audio CD collections, can be displayed in a separate window, which is shown in Figure 8.10. Several styles of visualizing the matches are possible. The figure shows the so-called “Timeline List View”. Here, each audio track that contains matches is displayed as an entry in a list. Each entry shows the title and meta information of the audio track and a set of green rectangular boxes on top of a blue bar indicating the position and duration of the matches on a timeline representing the audio track. The list is sorted by the ranking value of the top match contained in each entry. The ranking value is reflected in the transparency and color saturation of the rectangular boxes. Fully opaque and saturated green boxes, e.g., the first two boxes in the top row

of the illustration in Figure 8.10, indicate matches with very high similarity to the query. When hovering the mouse cursor over one of the matches, a pop-up frame displays additional information such as the temporal position of the region boundaries in the audio track and the ranking value. Clicking on a match opens the corresponding audio track in the audio viewer and starts playback from the starting position of the match. Furthermore, a corresponding sheet music book is displayed in the score viewer with the matches being highlighted as shown in the right part of Figure 8.9. In the top row of our example result list depicted in Figure 8.10, all four occurrences of the queried first theme of the Rondo of Beethoven's Piano Sonata No. 8, Op. 13, "Pathetique" have been found. Since the audio matching technique used for the content-based retrieval is capable of finding similarities across different interpretations of the same piece of music, and the audio database used in the example contains more than one interpretation of this piece, the following lines reveal the same four matches in different interpretations of the same piece.

## Chapter 9

# Conclusions

As a result of the work conducted in this thesis, we have seen that automatic bar-wise synchronization of scanned sheet music and audio recordings of Western classical music can be achieved. Furthermore, we have presented appealing end-user applications such as assisted score reading, visual navigation in audio recordings through the sheet music, switching of interpretations, and content-based search of sections in the score. However, in the scenario of building up a large-scale digital music library that offers synchronization between sheet music and audio recordings, several practical issues arise. The three main issues are the quality of the OMR results, the segmentation and identification of individual movements or songs in the sheet music and audio documents, as well as differences in the global structure between pairs of representations of the same movement or song. When restricting to certain types of music and levels of print quality, the issues can be solved in a way that minimizes the required manual interaction for organizing the data. Nevertheless, some issues still remain. In particular, large variations in tempo throughout a piece of music, transposing instruments, and special cases such as cadenzas, optional sections, missing sections, or alternative sections (COMA) may cause problems that require manual editing. Even though the approaches developed and discussed in this thesis seem promising and experiments deliver encouraging results, the system has yet to be put into practice in the targeted real-world environment to prove its usefulness and sustainability – a challenge that is currently being worked on.

This work has laid the foundation for many interesting future directions and further research. An obvious approach to encounter the issues of missing tempo information and tempo changes would be to identify and interpret the tempo directives found in the sheet music scans. Since the OMR results of SharpEye 2.68 are not reliable enough, one either has to incorporate other OMR software, or develop methods specifically tailored to the task that directly work with the scanned material itself. Here, one could incorporate knowledge about the expected position of tempo directives relative to the (already known) grand staves. A similar approach could be taken to encounter the issue of transposing instruments in orchestral scores. Here, what needs to be done basically is to determine which staves are played by which instrument group. A mapping between staves and instrument groups is usually indicated by text labels next to the first grand staff of the score of an orchestral piece, which could be identified and mapped to a known database of instrument group names and corresponding transpositions. However, the

task still includes many non-trivial problems to solve, such as individual instrument groups phasing out and back in the grand staffs off the orchestral score depending on whether or not they have notes to play during that part of the piece. Different conventions found in different scores on how to label the remaining instrument groups in such cases makes the task particularly hard.

To account for the missing information about important symbols and jump markers such as segnos and brackets for repeats with alternative endings, a straightforward approach would be to incorporate the results of a second OMR software that is capable of recognizing these symbols (e.g., SmartScore X). However, here one has to keep in mind that merging results from different OMR programs is far from trivial [71, 17]. Even though we were able to pragmatically fix some recognition errors of OMR using simple heuristics or sets of rules, a much better solution would be to incorporate more higher-level rules in the OMR software itself. Most OMR programs seem to make only very little use of the strong interdependencies that exist between the musical entities of sheet music. For example, the OMR engine used by SharpEye performs the recognition for each page individually. This means that any semantics that exceed the scope of a single page, e.g., key signatures and time signatures, can not be considered in the recognition and inference process of the immediately following pages. Most of the common OMR approaches follow a fixed processing pipeline to get from lower level entities to higher level entities. Therefore, high-level entities do not influence the decisions on the lower levels. To address this shortcoming, first mechanisms were introduced in [87] that allow higher-level steps to give feedback to lower-level steps. The decisions on how to group and construct the musical entities when reading a printed score are strongly interdependent. Using these interdependencies, a human reader is able to quickly recognize the musical entities and their semantic meaning. But when seen in an isolated fashion, as it is often done in optical music recognition systems, most of these decisions can only be made with high uncertainty.

A very interesting but also very challenging direction of future work would be to find methods for automatically detecting COMA cases such as cadenzas. For simpler cases such as locating the boundaries of a cadenza in an audio recording for which the position in the score is known, this seems quite feasible. More general and advanced cases such as handling cadenzas or missing sections for which there is no evidence in the score data, however, might first require improvements in other issues like OMR quality and tempo estimation before a reliable automatic detection becomes possible. To encounter issues with differences in tempo, an interesting strategy might be to extend the concept of mid-level features for local content comparison to incorporate independency of tempo (or at least robustness against temporal differences to a high degree). From symbolic note events such tempo-independent features could easily be derived by simply assigning one temporal frame to each time position where there is at least one note onset. However, deriving such features from audio recordings seems much harder. Possible approaches would be to incorporate transcription or tempo estimation/beat tracking techniques like [70, 35, 59] to eliminate differences and variations in tempo. Unfortunately, especially for classical music, the results that can be obtained with these techniques may not be reliable enough to yield a lot of improvement at the present time.

In a different direction of future work, one might work towards making the sheet music-audio synchronization finer and more robust. Using techniques for improving the temporal resolution of the synchronization such as [43] one can probably achieve a reasonable accuracy

on a sub-bar level such as individual notes. The chroma features, cost/similarity measures and DTW settings used in the synchronization process include a variety of parameters that could be tuned and optimized to achieve a higher robustness. Here, it would be especially interesting to measure the effect of different settings and OMR errors on the overall synchronization results. One might also try to replace the DTW approach with an approach based on HMMs to allow finding optimal parameters through machine learning based on training data.

As already discussed in Chapter 6, important future work would be to implement the string-based comparison of titles and headings to assist in the task of track segmentation and identification. An approach to improving the partial synchronization of sheet music and audio discussed in Chapter 7 that might be worth pursuing is to incorporate automatic structure analysis of audio recordings as an additional source of information. Finally, we want to point out that a successful sheet music-audio synchronization paves the way for more applications than those presented in Chapter 8. Here, a very interesting idea is to use the symbolic data that is available for each staff individually to perform a time-dependent staff-based EQ-ing or filtering to acoustically highlight or suppress individual instrument groups in multi-instrument recordings, as has already been proposed in [107].



# Appendix A

## Calculations

In this Section, we give the details on the calculation of the computational complexity for the iterative approach for reducing inconsistency described in Section 5.3. Note that the calculation presented here is just a rough heuristic estimate rather than a detailed analysis. Since we expect the predominant factor in the computational cost of our algorithm to be checking the data for rule violations when evaluating the inconsistency measure, we want to estimate how many of these evaluations to expect depending on the length of the data and the number of rules included in the rule set. Let  $B$  be the number of bars in the given OMR data and  $R$  be the number of rules included in the rule set. We assume that the cost  $t_{\text{rule}}$  for checking the data for violations w.r.t. a single rule scales linearly with the length of the data, i.e.,  $t_{\text{rule}} = c_1 \cdot B$  for some constant  $c_1$ . For a single evaluation of the inconsistency measure we need to check the data against a total of  $R$  rules resulting in a computational cost

$$t_{\text{eval}} = R \cdot t_{\text{rule}} = c_1 \cdot B \cdot R. \quad (\text{A.1})$$

Let  $V(k) = |V^k|$  be the number of violations obtained at the  $k$ -th iteration. To simplify our calculation, we assume that we start with  $V(0) = K$  violations at the 0-th iteration and eliminate one violation with each iteration until no more violation is left after a total of  $K$  iterations

$$V(k) = K - k, \quad k = 0, \dots, K. \quad (\text{A.2})$$

Note that this assumption usually does not hold in practice, because a single modification to the data can solve many violations at once but can also cause several new violations. However, it does seem reasonable to assume that, on the average,  $V(k)$  linearly decreases with  $k$ . We approximate the number of candidate modifications obtained at the  $k$ -th iteration  $M(k) = |M^k|$  as being proportional  $V(k)$  by setting

$$M(k) = c_2 \cdot V(k) \quad (\text{A.3})$$

for some proportionality constant  $c_2$ . In each iteration, we perform one evaluation of the inconsistency measure on  $\Omega^k$  in Steps 2 to 3 of the algorithm, and perform  $M(k)$  evaluations on the modified data  $\Omega_m^k$  in Steps 6 to 7. Therefore, for the  $k$ -th iteration, we perform

$1 + M(k)$  evaluations of the inconsistency measure. The total computational cost for all  $K$  iterations can then be estimated as

$$\begin{aligned}
t_{\text{total}} &= \sum_{k=0}^K (1 + M(k)) \cdot t_{\text{eval}} \\
&= \left( (K + 1) + \sum_{k=0}^K M(k) \right) \cdot t_{\text{eval}} \\
&= \left( 1 + K + \sum_{k=0}^K c_2 \cdot V(k) \right) \cdot t_{\text{eval}} \\
&= \left( 1 + K + c_2 \sum_{k=0}^K (K - k) \right) \cdot t_{\text{eval}}. \tag{A.4}
\end{aligned}$$

The sum in Equation A.4 can be calculated as

$$\begin{aligned}
\sum_{k=0}^K (K - k) &= \sum_{k=0}^K k \\
&= \frac{K \cdot (K + 1)}{2} \\
&= \frac{1}{2}(K^2 + K).
\end{aligned}$$

Replacing the sum in Equation A.4 with this result, we get

$$\begin{aligned}
t_{\text{total}} &= \left( 1 + K + \frac{1}{2}c_2(K^2 + K) \right) \cdot t_{\text{eval}} \\
&= \left( 1 + \left(1 + \frac{1}{2}c_2\right)K + \frac{1}{2}c_2K^2 \right) \cdot t_{\text{eval}}. \tag{A.5}
\end{aligned}$$

It is reasonable to assume that the amount of violations  $K = V(0)$  after 0 iterations scales linearly with both the number of bars  $B$  and the number of rules  $R$ . Therefore, we set

$$K = c_3 \cdot B \cdot R \tag{A.6}$$

for some constant  $c_3$ . Using Equations (A.1) and (A.6) in Equation (A.5), we get

$$\begin{aligned}
t_{\text{total}} &= \left( 1 + \left(1 + \frac{1}{2}c_2\right)K + \frac{1}{2}c_2K^2 \right) \cdot t_{\text{eval}} \\
&= \left( 1 + \left(1 + \frac{1}{2}c_2\right)c_3 \cdot B \cdot R + \frac{1}{2}c_2c_3^2 \cdot B^2 \cdot R^2 \right) \cdot c_1 \cdot B \cdot R \\
&= c_1BR + \left(1 + \frac{1}{2}c_2\right)c_1c_3 \cdot B^2 \cdot R^2 + \frac{1}{2}c_1c_2c_3^2 \cdot B^3 \cdot R^3 \\
&= \mathcal{O}(B^3 \cdot R^3).
\end{aligned}$$



# Bibliography

- [1] Vlora Arifi, Michael Clausen, Frank Kurth, and Meinard Müller. Synchronization of music data in score-, MIDI- and PCM-format. *Computing in Musicology*, 13:9–33, 2004.
- [2] Andreas Arzt, Gerhard Widmer, and Simon Dixon. Automatic page turning for musicians via Real-Time machine listening. In *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI 2008)*, Patras, Greece, 2008.
- [3] Denis Baggi, Adriano Barate, Goffredo Haus, and Luca Andrea Ludovico. NINA – navigating and interacting with notation and audio. In *Proceedings of the second International Workshop on Semantic Media Adaptation and Personalization (SMAP)*, pages 134–139, Uxbridge, Middlesex, UK, 2007.
- [4] David Bainbridge and Tim Bell. The challenge of optical music recognition. *Computers and the Humanities*, 35(2):95–121, 2001.
- [5] David Bainbridge and Tim Bell. A music notation construction engine for optical music recognition. *Software – Practice & Experience*, 33(2):173–200, 2003.
- [6] Adriano Baratè and Luca A. Ludovico. Advanced interfaces for music enjoyment. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, pages 421–424, Napoli, Italy, 2008.
- [7] Mark A. Bartsch and Gregory H. Wakefield. To Catch a Chorus: Using chroma-based representations for audio thumbnailing. In *Proceedings of the IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, pages 15–18, New Paltz, NY, USA, 2001.
- [8] Mark A. Bartsch and Gregory H. Wakefield. Audio thumbnailing of popular music using chroma-based representations. *IEEE Transactions on Multimedia*, 7(1):96–104, 2005.
- [9] Pierfrancesco Bellini, Ivan Bruno, and Paolo Nesi. Optical music sheet segmentation. In *Proceedings of the 1st International Conference on Web Delivering of Music*, 2001.
- [10] Pierfrancesco Bellini, Ivan Bruno, and Paolo Nesi. Assessing optical music recognition tools. *Computer Music Journal*, 31(1):68–93, 2007.
- [11] Juan Pablo Bello and Jeremy Pickens. A robust mid-level representation for harmonic content in music signals. In *Proceedings of the 6th International Conference on Music Information Retrieval (ISMIR)*, London, UK, 2005.

- [12] Nancy Bertin, Roland Badeau, and Emmanuel Vincent. Fast Bayesian NMF algorithms enforcing harmonicity and temporal continuity in polyphonic music transcription. In *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, New Paltz, New York, USA, 2009.
- [13] Ina Blümel, Jürgen Diet, and Harald Krottmaier. Integrating multimedia repositories into the PROBADO framework. In *Proceedings of the 3rd International Conference on Digital Information Management (ICDIM)*, Michigan, USA, 2008.
- [14] Donald Byrd. Music notation software and intelligence. *Computer Music Journal*, 18(1):17–20, 1994.
- [15] Donald Byrd. Written vs. sounding pitch. *MLA Notes*, 66(1), 2009.
- [16] Donald Byrd and Tim Crawford. Problems of music information retrieval in the real world. *Information Processing and Management: an International Journal*, 38(2):249–272, 2002.
- [17] Donald Byrd, William Guerin, Megan Schindele, and Ian Knopke. OMR evaluation and prospects for improved OMR via multiple recognizers. Technical report, Indiana University, Bloomington, Indiana, USA, 2010.
- [18] Donald Byrd and Megan Schindele. Prospects for improving OMR with multiple recognizers. In *Proceedings of the 7th International Conference on Music Information Retrieval (ISMIR)*, pages 41–46, 2006.
- [19] Pedro Cano, Alex Liscos, and Jordi Bonada. Score-performance matching using HMMs. In *Proceedings of the International Computer Music Conference (ICMC)*, 1999.
- [20] Michael Casey, Remco Veltkamp, Masataka Goto, Marc Leman, Christopher Rhodes, and Malcolm Slaney. Content-based music information retrieval: Current directions and future challenges. *Proceedings of the IEEE*, 96(4):668–696, 2008.
- [21] Wei Chai. Semantic segmentation and summarization of music: Methods based on tonality and recurrent structure. *IEEE Signal Processing Magazine*, 23(2):124–132, 2006.
- [22] Kai Chen, Sheng Gao, Yongwei Zhu, and Qibin Sun. Popular song and lyrics synchronization and its application to music information retrieval. In *Proceedings of the 13th Annual Multimedia Computing and Networking Conference*, San Jose, CA, USA, 2006.
- [23] G. Sayeed Choudhury, Tim DiLauro, Michael Droettboom, Ichiro Fujinaga, Brian Harrington, and Karl MacMillan. Optical music recognition system within a large-scale digitization project. In *Proceedings of the 1st International Symposium on Music Information Retrieval (ISMIR)*, Plymouth, MA, USA, 2000.
- [24] Arshia Cont. Realtime audio to score alignment for polyphonic music instruments, using sparse non-negative constraints and hierarchical HMMs. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 5, 2006.

- [25] Arshia Cont. ANTESCOFO: Anticipatory synchronization and control of interactive parameters in computer music. In *Proceedings of International Computer Music Conference (ICMC)*, Belfast, Ireland, 2008.
- [26] Arshia Cont. A coupled duration-focused architecture for realtime music to score alignment. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 32(6):974–987, 2010.
- [27] David Damm, Christian Fremerey, Frank Kurth, Meinard Müller, and Michael Clausen. Multimodal presentation and browsing of music. In *Proceedings of the 10th International Conference on Multimodal Interfaces (ICMI)*, Chania, Crete, Greece, 2008.
- [28] Roger Dannenberg and Ning Hu. Polyphonic audio matching for score following and intelligent audio editors. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 27–34, San Francisco, USA, 2003.
- [29] Roger B. Dannenberg. An on-line algorithm for real-time accompaniment. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 193–98, Paris, France, 1984.
- [30] Roger B. Dannenberg and Christopher Raphael. Music score alignment and computer accompaniment. *Communications of the ACM, Special Issue: Music information retrieval*, 49(8):38–43, 2006.
- [31] Peter Desain, Henkjan Honing, and Hank Heijink. Robust score-performance matching: Taking advantage of structural information. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 337–40, 1997.
- [32] Johanna Devaney and Daniel P.W. Ellis. Handling asynchrony in audio-score alignment. In *Proceedings of the International Computer Music Conference (ICMC)*, Montreal, 2009.
- [33] Johanna Devaney, Michael I. Mandel, and Daniel P.W. Ellis. Improving MIDI-audio alignment with acoustic features. In *Proceedings of the IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, pages 45–48, Mohonk, NY, 2009.
- [34] Jürgen Diet and Frank Kurth. The PROBADO music repository at the Bavarian State Library. In *Proceedings of the 8th International Conference on Music Information Retrieval (ISMIR)*, pages 501–504, Vienna, Austria, 2007.
- [35] Simon Dixon. Evaluation of the audio beat tracking system BeatRoot. *Journal of New Music Research*, 36:39–50, 2007.
- [36] Simon Dixon and Gerhard Widmer. MATCH: A music alignment tool chest. In *Proceedings of the 6th International Conference on Music Information Retrieval (ISMIR)*, pages 492–497, London, UK, 2005.
- [37] Michael Droettboom and Ichiro Fujinaga. Symbol-Level groundtruthing environment for OMR. In *Proceedings of the 5th International Conference on Music Information Retrieval (ISMIR)*, Barcelona, Spain, 2004.

- [38] Michael Droettboom, Ichiro Fujinaga, and Karl MacMillan. Optical music interpretation. In *Proceedings of the Joint IAPR International Workshop on Structural, Syntactic, and Statistical Pattern Recognition*, pages 378–386, 2002.
- [39] Michael Droettboom, Karl MacMillan, and Ichiro Fujinaga. The gamera framework for building custom recognition systems. In *Symposium on Document Image Understanding Technologies*, pages 275–286, 2003.
- [40] Jon W. Dunn, Donald Byrd, Mark Notess, Jenn Riley, and Ryan Scherle. Variations2: Retrieving and using music in an academic setting. *Communications of the ACM, Special Issue: Music information retrieval*, 49(8):53–48, 2006.
- [41] Richard Durbin, Sean R. Eddy, Anders Krogh, and Graeme Mitchison. *Biological sequence analysis: Probabilistic models of proteins and nucleic acids*. Cambridge University Press, 1999.
- [42] Sebastian Ewert and Meinard Müller. Refinement strategies for music synchronization. In *Proceedings of the 5th International Symposium on Computer Music Modeling and Retrieval (CMMR)*, Copenhagen, Denmark, 2008.
- [43] Sebastian Ewert, Meinard Müller, and Peter Grosche. High resolution audio synchronization using chroma onset features. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Taipei, Taiwan, 2009.
- [44] Christian Fremerey. SyncPlayer – a framework for content-based music navigation. Diploma Thesis, Department of Computer Science III, University of Bonn, 2006.
- [45] Christian Fremerey, Michael Clausen, Meinard Müller, and Sebastian Ewert. Sheet music-audio identification. In *Proceedings of the 10th International Conference on Music Information Retrieval (ISMIR)*, pages 645–650, Kobe, Japan, 2009.
- [46] Christian Fremerey, Meinard Müller, and Michael Clausen. Towards bridging the gap between sheet music and audio. In Eleanor Selfridge-Field, Frans Wiering, and Geraint A. Wiggins, editors, *Knowledge representation for intelligent music processing*, Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2009. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany.
- [47] Christian Fremerey, Meinard Müller, and Michael Clausen. Handling repeats and jumps in score-performance synchronization. In *(to appear) Proceedings of the 11th International Society for Music Information Retrieval Conference (ISMIR)*, Utrecht, Netherlands, 2010.
- [48] Christian Fremerey, Meinard Müller, Frank Kurth, and Michael Clausen. Automatic mapping of scanned sheet music to audio recordings. In *Proceedings of the 9th International Conference on Music Information Retrieval (ISMIR)*, pages 413–418, Philadelphia, USA, 2008.
- [49] Hiromasa Fujihara, Masataka Goto, Jun Ogata, Kazunori Komatani, Tetsuya Ogata, and Hiroshi G. Okuno. Automatic synchronization between lyrics and music CD recordings based on Viterbi alignment of segregated vocal signals. In *Proceedings of the Eighth IEEE International Symposium on Multimedia*, pages 257–264, 2006.

- [50] Ichiro Fujinaga. *Adaptive Optical Music Recognition*. Ph.D dissertation, McGill University, Montreal, Canada, 1996.
- [51] Takuya Fujishima. Real-time chord recognition of musical sound: a system using Common Lisp Music. In *Proceedings of the International Computer Music Conference (ICMC)*, Beijing, China, 1999.
- [52] Jörg Garbers and Frans Wiering. Towards structural alignment of folk songs. In *Proceedings of the 9th International Conference on Music Information Retrieval (ISMIR)*, pages 381–386, Philadelphia, USA, 2008.
- [53] Thilo Geertzen. Ansätze zur Behandlung von Strukturunterschieden und Sprungstellen bei der Synchronisation von Notenscans und Audioaufnahmen. Diploma thesis, Department of Computer Science III, University of Bonn, 2009.
- [54] Emilia Gomez. *Tonal description of music audio signals*. Ph.D dissertation, Music Technology Group, University Pompeu Fabra, Barcelona, Spain, 2006.
- [55] Michael Good. MusicXML: An internet-friendly format for sheet music. In *Proceedings of XML*, Boston, USA, 2001.
- [56] Masataka Goto. A chorus-section detecting method for musical audio signals. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 437–440, Hong Kong, China, 2003.
- [57] Masataka Goto. SmartMusicKIOSK: Music listening station with chorus-search function. In *Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology (UIST)*, pages 31–40, 2003.
- [58] Masataka Goto, Hiroki Hashiguchi, Takuichi Nishimura, and Ryuichi Oka. RWC music database: Popular, classical and jazz music databases. In *Proceedings of the 3rd International Conference on Music Information Retrieval (ISMIR)*, 2002.
- [59] Peter Grosche and Meinard Müller. Computing predominant local periodicity information in music recordings. In *Proceedings of the IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, New Paltz, New York, USA, 2009.
- [60] Lorin Grubb and Roger B. Dannenberg. A stochastic method for tracking a vocal performer. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 301–308, 1997.
- [61] Lorin Grubb and Roger B. Dannenberg. Enhanced vocal performance tracking using multiple information sources. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 37–44, 1998.
- [62] Andrew Hankinson, Laurent Pugin, and Ichiro Fujinaga. Interfaces for document representation in digital music libraries. In *Proceedings of the 10th International Conference on Music Information Retrieval (ISMIR)*, Kobe, Japan, 2009.

- [63] Brian Harrington, James W. Warner, Elizabeth W. Brown, Tim DiLauro, Ichiro Fujinaga, Cynthia Requardt, and G. Sayeed Choudhury. Digital workflow management: The Lester S. Levy digitized collection of sheet music. *First Monday*, 5(6), 2000.
- [64] Hank Heijink, Peter Desain, Henkjan Honing, and L. Windsor. Make Me a Match: An evaluation of different approaches to score-performance matching. *Computer Music Journal*, 24(1):43–56, 2000.
- [65] Walter Hewlett. MuseData – an electronic library of classical music scores. Center for Computer Assisted Research in the Humanities, <http://www.musedata.org>, Accessed: April 2010.
- [66] Ning Hu, Roger B. Dannenberg, and George Tzanetakis. Polyphonic audio matching and alignment for music retrieval. In *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, pages 185–188, New York, 2003.
- [67] David Huron. Music information processing using the Humdrum Toolkit: Concepts, examples, and lessons. *Computer Music Journal*, 26(2):15–30, 2002.
- [68] IFLA Study Group on the Functional Requirements of Bibliographic Records. Functional Requirements for Bibliographic Records; Final Report. Saur, Munich, 1998. available at [www.ifla.org/VII/s13/frbr/frbr.pdf](http://www.ifla.org/VII/s13/frbr/frbr.pdf).
- [69] Kent Kennan and Donald Grantham. *The Technique of Orchestration*. Prentice Hall, 6th edition, 2002.
- [70] Anssi Klapuri and Manuel Davy. *Signal processing methods for music transcription*. Springer, 2006.
- [71] Ian Knopke and Donald Byrd. Towards Musicdiff: A foundation for improved optical music recognition using multiple recognizers. In *Proceedings of the 8th International Conference on Music Information Retrieval (ISMIR)*, pages 123–126, Vienna, Austria, 2007.
- [72] Harald Krottmaier, Frank Kurth, Thorsten Steenweg, Hans-Jürgen Apperath, and Dieter Fellner. PROBADO - a generic repository integration framework. In *Proceedings of the 11th European Conference on Digital Libraries (ECDL)*, Budapest, Hungary, 2007.
- [73] Frank Kurth, David Damm, Christian Fremerey, Meinard Müller, and Michael Clausen. A framework for managing multimodal digitized music collections. In *Proceedings of the 12th European Conference on Research and Advanced Technology for Digital Libraries (ECDL)*, Aarhus, Denmark, 2008.
- [74] Frank Kurth and Meinard Müller. Efficient index-based audio matching. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(2):382–395, 2008.
- [75] Frank Kurth and Meinard Müller. Efficient index-based audio matching. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(2):382–395, 2008.

- [76] Frank Kurth, Meinard Müller, David Damm, Christian Fremerey, Andreas Ribbrock, and Michael Clausen. SyncPlayer - an advanced system for content-based audio access. In *Proceedings of the 6th International Conference on Music Information Retrieval (ISMIR)*, pages 381–388, London, GB, 2005.
- [77] Frank Kurth, Meinard Müller, and Christian Fremerey. Audio Matching für symbolische Musikdaten. In *Fortschritte der Akustik, Tagungsband der DAGA*, 2007.
- [78] Frank Kurth, Meinard Müller, Christian Fremerey, Yoon ha Chang, and Michael Clausen. Automated synchronization of scanned sheet music with audio recordings. In *Proceedings of the 8th International Conference on Music Information Retrieval (ISMIR)*, pages 261–266, Vienna, Austria, 2007.
- [79] Christian Landone, Joseph Harrop, and Joshua D. Reiss. Enabling access to sound archives through integration, enrichment and retrieval: The EASAIER project. In *Proceedings of the 8th International Conference on Music Information Retrieval (ISMIR)*, pages 159–160, Vienna, Austria, 2007.
- [80] John Lawter and Barry Moon. Score following in open form compositions. In *Proceedings of the International Computer Music Conference (ICMC)*, 1998.
- [81] Kyogu Lee and Markus Cremer. Segmentation-based lyrics-audio alignment using dynamic programming. In *Proceedings of the 9th International Conference on Music Information Retrieval (ISMIR)*, pages 395–400, Philadelphia, USA, 2008.
- [82] Mark Levy and Mark Sandler. Structural segmentation of musical audio by constrained clustering. *IEEE Transactions on Audio, Speech and Language Processing*, 16(2):318–326, 2008.
- [83] Classical Archives LLC. Classical Archives – The Ultimate Classical Music Destination. <http://www.classicalarchives.com>, Accessed: April 2010.
- [84] Karl MacMillan, Michael Droettboom, and Ichiro Fujinaga. Gamera: Optical music recognition in a new shell. In *Proceedings of the International Computer Music Conference (ICMC)*, 2002.
- [85] Namunu C. Maddage, Changsheng Xu, Mohan S. Kankanhalli, and Xi Shao. Content-based music structure analysis with applications to music semantics understanding. In *Proceedings of the ACM Multimedia*, pages 112–119, New York, NY, USA, 2004.
- [86] Stephen Malinowski. The Music Animation Machine. <http://www.musanim.com>, 1988, Accessed: April 2010.
- [87] John McPherson. *Coordinating Knowledge to Improve Optical Music Recognition*. Ph.D dissertation, The University of Waikato, 2006.
- [88] MIDI. Musical Instrument Digital Interface. Manufacturers Association, <http://www.midi.org>, Accessed: 2006.
- [89] Riccardo Miotto and Nicola Orio. Automatic identification of music works through audio matching. In *Proceedings of the 11th European Conference on Digital Libraries (ECDL)*, pages 124–135, Budapest, Hungary, 2007.

- [90] Nicola Montecchio and Nicola Orio. Automatic alignment of music performances with scores aimed at educational applications. In *Proceedings of the International Conference on Automated Solutions for Cross Media Content and Multi-Channel Distribution*, pages 17–24, 2008.
- [91] Meinard Müller. *Information Retrieval for Music and Motion*. Springer, 2007.
- [92] Meinard Müller and Daniel Appelt. Path-constrained partial music synchronization. In *Proceedings of the 34th International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 1, pages 65–68, Las Vegas, Nevada, USA, 2008.
- [93] Meinard Müller, Michael Clausen, Verena Konz, Sebastian Ewert, and Christian Fremerey. A multimodal way of experiencing and exploring music. (*to appear*) *Interdisciplinary Science Reviews (ISR), Special Issue: Music and the Sciences*, 35(2), 2010.
- [94] Meinard Müller and Sebastian Ewert. Joint structure analysis with applications to music annotation and synchronization. In *Proceedings of the 9th International Conference on Music Information Retrieval (ISMIR)*, pages 389–394, Philadelphia, USA, 2008.
- [95] Meinard Müller and Sebastian Ewert. Towards timbre-invariant audio features for harmony-based music. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(3):649–662, 2010.
- [96] Meinard Müller and Frank Kurth. Towards structural analysis of audio recordings in the presence of musical variations. *EURASIP Journal on Advances in Signal Processing*, 2007(1), 2007.
- [97] Meinard Müller, Frank Kurth, and Michael Clausen. Audio matching via chroma-based statistical features. In *Proceedings of the 6th International Conference on Music Information Retrieval (ISMIR)*, pages 288–295, London, GB, 2005.
- [98] Meinard Müller, Frank Kurth, David Damm, Christian Fremerey, and Michael Clausen. Lyrics-based audio retrieval and multimodal navigation in music collections. In *Proceedings of the 11th European Conference on Digital Libraries (ECDL)*, Budapest, Hungary, 2007.
- [99] Meinard Müller, Frank Kurth, and Tido Röder. Towards an efficient algorithm for automatic score-to-audio synchronization. In *Proceedings of the 5th International Conference on Music Information Retrieval (ISMIR)*, pages 365–372, Barcelona, Spain, 2004.
- [100] Meinard Müller, Hennig Mattes, and Frank Kurth. An efficient multiscale approach to audio synchronization. In *Proceedings of the 7th International Conference on Music Information Retrieval (ISMIR)*, pages 192–197, Victoria, Canada, 2006.
- [101] Myriad Software. PDFtoMusic Pro. <http://www.myriad-online.com>, Accessed: April 2010.
- [102] Bernhard Niedermayer. Improving accuracy of polyphonic music-to-score alignment. In *Proceedings of the 10th International Conference on Music Information Retrieval (ISMIR)*, pages 645–650, Kobe, Japan, 2009.



- [103] Bernhard Niedermayer. Towards audio to score alignment in the symbolic domain. In *Proceedings of the Sound and Music Computing Conference (SMC 2009)*, Porto, Portugal, 2009.
- [104] Han-Wen Nienhuys and Jan Nieuwenhuizen. Lilypond, a system for automated music engraving. In *Proceedings of the 14th Colloquium on Musical Informatics (CIM)*, Firenze, Italy, 2003.
- [105] Nicola Orio. Alignment of performances with scores aimed at content-based music access and retrieval. In *Proceedings of the 6th European Conference on Research and Advanced Technology for Digital Libraries (ECDL)*, pages 479–492, Rome, Italy, 2002.
- [106] Nicola Orio. A system for the automatic identification of music works. In *Proceedings of the 14th International Conference of Image Analysis and Processing – Workshops (ICIAPW)*, pages 15–20, Modena, Italy, 2007.
- [107] Nicola Orio. A discrete filter bank approach to audio to score matching for polyphonic music. In *Proceedings of the 10th International Conference on Music Information Retrieval (ISMIR)*, Kobe, Japan, 2009.
- [108] Nicola Orio and Francois Dechelle. Score following using spectral analysis and hidden Markov models. In *Proceedings of the International Computer Music Conference (ICMC)*, 2001.
- [109] Nicola Orio, Serge Lemouton, and Diemo Schwarz. Score following: State of the art and new developments. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*, 2003.
- [110] Nicola Orio and Diemo Schwarz. Alignment of monophonic and polyphonic music to a score. *Proceedings of the International Computer Music Conference (ICMC)*, pages 155–158, 2001.
- [111] Bryan Pardo and William Birmingham. Modeling form for on-line following of musical performances. In *Proceedings of the 20th National Conference on Artificial Intelligence*, Pittsburgh, Pennsylvania, 2005.
- [112] Bryan Pardo and William P. Birmingham. Following a musical performance from a partially specified score. In *Proceedings of the Multimedia Technology and Applications Conference*, Irvine, California, 2001.
- [113] Jeremy Pickens, Juan Pablo Bello, Giuliano Monti, Tim Crawford, Matthew Dovey, and Mark Sandler. Polyphonic score retrieval using polyphonic audio queries: A harmonic modeling approach. In *Proceedings of the 3rd International Conference on Music Information Retrieval (ISMIR)*, pages 140–149, Paris, France, 2002.
- [114] Miller Puckette. Score following using the sung voice. In *Proceedings of the International Computer Music Conference (ICMC)*, 1995.
- [115] Laurent Pugin, John Ashley Burgoyne, and Ichiro Fujinaga. Goal-directed evaluation for the improvement of optical music recognition on early music prints. In *Proceedings of the ACM-IEEE Joint Conference on Digital Libraries (JCDL)*, pages 303–304, Vancouver, British Columbia, 2007.

- [116] Laurent Pugin, John Ashley Burgoyne, and Ichiro Fujinaga. MAP adaptation to improve optical music recognition of early music documents using hidden Markov models. In *Proceedings of the 8th International Conference on Music Information Retrieval (ISMIR)*, pages 513–516, Vienna, Austria, 2007.
- [117] Laurent Pugin, Jason Hockman, John Ashley Burgoyne, and Ichiro Fujinaga. Gamera versus Aruspix: Two optical music recognition approaches. In *Proceedings of the 9th International Conference on Music Information Retrieval (ISMIR)*, pages 419–424, Philadelphia, USA, 2008.
- [118] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.
- [119] Christopher Raphael. Automatic segmentation of acoustic musical signals using hidden Markov models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(4):360–370, 1999.
- [120] Christopher Raphael. A Bayesian network for real-time musical accompaniment. *Neural Information Processing Systems (NIPS)*, 14, 2001.
- [121] Christopher Raphael. Music Plus One: A system for flexible and expressive musical accompaniment. In *Proceedings of the International Computer Music Conference (ICMC)*, Havana, Cuba, 2001.
- [122] Christopher Raphael. Orchestral musical accompaniment from synthesized audio. In *Proceedings of the International Computer Music Conference (ICMC)*, 2003.
- [123] Christopher Raphael. A hybrid graphical model for aligning polyphonic audio with musical scores. In *Proceedings of the 5th International Conference on Music Information Retrieval (ISMIR)*, Barcelona, Spain, 2004.
- [124] Christopher Raphael. Musical accompaniment systems. *Chance Magazine*, 17(4):17–22, 2004.
- [125] Christopher Raphael. Aligning music audio with symbolic scores using a hybrid graphical model. *Machine Learning*, 65(2-3):389–409, 2006.
- [126] Florence Rossant and Isabelle Bloch. A fuzzy model for optical recognition of musical scores. *Fuzzy Sets and Systems*, 141(2):165–201, 2004.
- [127] Shoichiro Saito, Hirokazu Kameoka, Keigo Takahashi, Takuya Nishimoto, and Shigeki Sagayama. Specmurt analysis of polyphonic music signals. *IEEE Transactions on Audio, Speech and Language Processing*, 16(3):639–650, 2008.
- [128] Stan Salvador and Philip Chan. Toward accurate dynamic time warping in linear time and space. *Intelligent Data Analysis*, 11(5):561–580, 2007.
- [129] Craig S. Sapp. Online database of scores in the Humdrum file format. In *Proceedings of the 6th International Conference on Music Information Retrieval (ISMIR)*, London, GB, 2005.

- [130] Craig S. Sapp. Comparative analysis of multiple musical performances. In *Proceedings of the 8th International Conference on Music Information Retrieval (ISMIR)*, Vienna, Austria, 2007.
- [131] Diemo Schwarz, Arshia Cont, and Norbert Schnell. From Boulez to Ballads: Training IRCAM's score follower. In *Proceedings of the International Computer Music Conference (ICMC)*, Barcelona, Spain, 2005.
- [132] Eleanor Selfridge-Field, editor. *Beyond MIDI: The Handbook of Musical Codes*. MIT Press, Cambridge, MA, USA, 1997.
- [133] Joan Serrà, Emilia Gomez, Perfecto Herrera, and Xavier Serra. Chroma binary similarity and local alignment applied to cover song identification. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(6):1138–1151, 2008.
- [134] Leland A. Smith. The SCORE music publishing system. San Andreas Press, <http://www.scoremus.com>, Accessed: April 2010.
- [135] Ferreol Soulez, Xavier Rodet, and Diemo Schwarz. Improving polyphonic and poly-instrumental music to score alignment. In *Proceedings of the 4th International Conference on Music Information Retrieval (ISMIR)*, Baltimore, Maryland, USA, 2003.
- [136] Thorsten Steenweg and Ulrike Steffens. PROBADO – non-textual digital libraries put into practice. *ERCIM News, Special Theme: European Digital Library*, pages 47–48, 2006.
- [137] Imam S.H. Suyoto, Alexandra L. Uitdenbogerd, and Falk Scholer. Searching musical audio using symbolic queries. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(2):372–381, 2008.
- [138] Iman S.H. Suyoto, Alexandra L. Uitdenbogerd, and Falk Scholer. Effective retrieval of polyphonic audio with polyphonic symbolic queries. In *Proceedings of the International Workshop on Multimedia Information Retrieval*, pages 105–114, Augsburg, Bavaria, Germany, 2007.
- [139] Mariusz Szwoch. Guido: A musical score recognition system. In *Proceedings of the 9th International Conference on Document Analysis and Recognition (ICDAR)*, volume 2, pages 809–813, 2007.
- [140] Mevlut Evren Tekin, Christina Anagnostopoulou, and Yo Tomita. Towards an intelligent score following system: Handling of mistakes and jumps encountered during piano practicing. In *Computer Music Modeling and Retrieval*, pages 211–219. 2005.
- [141] Verena Thomas, Christian Fremerey, David Damm, and Michael Clausen. SLAVE: A Score-Lyrics-Audio-Video-Explorer. In *Proceedings of the 10th International Conference on Music Information Retrieval (ISMIR)*, Kobe, Japan, 2009.
- [142] Robert J. Turetsky and Daniel P.W. Ellis. Ground-truth transcriptions of real music from force-aligned MIDI syntheses. In *Proceedings of the 4th International Conference on Music Information Retrieval (ISMIR)*, pages 135–141, Baltimore, Maryland, USA, 2003.

- [143] Barry Vercoe. The synthetic performer in the context of live performance. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 199–200, Paris, France, 1984.
- [144] Verein Arbeitsstelle Schweiz des RISM. Informationspool “Repertory of 19th century’s swiss composers”. <http://www.rism-ch.org/pages/informationpool>, Accessed: April 2010.
- [145] Ye Wang, Min-Yen Kan, Tin Lay Nwe, Arun Shenoy, and Jun Yin. LyricAlly: automatic synchronization of acoustic musical signals and textual lyrics. In *Proceedings of the 12th Annual ACM international conference on Multimedia*, pages 212–219, New York, NY, USA, 2004.
- [146] Jan Weil, Thomas Sikora, J.-L. Durrieu, and Gaël Richard. Automatic generation of lead sheets from polyphonic music signals. In *Proceedings of the 10th International Conference on Music Information Retrieval (ISMIR)*, pages 603–608, Kobe, Japan, 2009.
- [147] Joe Wolfe. Note names, MIDI numbers and frequencies. <http://www.phys.unsw.edu.au/jw/notes.html>, Accessed: March 2010.
- [148] Yongwei Zhu and M.S. Kankanhalli. Precise Pitch Profile feature extraction from musical audio for key detection. *IEEE Transactions on Multimedia*, 8(3):575–584, 2006.

# Index

- accumulated cost matrix, 29
- alignment, *see* synchronization
- alignment path, 27
  - optimum, 28
- audio, 9
  - matching, 150
  - structure analysis, 13
  - thumbnailing, 13
  - track, 50
  - viewer, 146
- automatic computer accompaniment, 9
- bar, 25
  - source, 121
  - target, 121
- bitmap data, 17
- block, 121
  - boundaries, 121
  - boundary indicators, 122
  - feature, 124
- chord, 71
- chroma, 39
  - features, 39
- COMA, 62
  - classes, 62
  - entries, 62
- connections, 131
  - jump, 132
  - regular, 131
- content-based retrieval, 12
- cover song identification, 13
- cross-domain, 148
- da capo, 88
- dal segno, 88
- database, 40
- default bar sequence, 25
- degeneration, 126
  - horizontal, 127
  - vertical, 127
- dynamic time warping, 29
  - jump, 130
- engraving data, 17
- execution, 18
- expected performance block index sequence, 124
- expression, 44
- feature blocks, 124
- fine, 88
- frames, 27
- geometric music reading, 18
- item, 44
- jump, 121
  - backward, 121
  - external, 121
  - forward, 121
  - internal, 121
- JumpDTW, 130
- leaf-level
  - track, 50
  - work, 44
- local cost
  - matrix, 27, 29
  - measure, 27
- manifestation, 44
  - reference, 59
- match, 12, 41
- matching, 40
  - audio, 150
  - curve, 40
- matrix
  - accumulated cost, 29
  - local cost, 27, 29

- optimum predecessor position, 29
- mid-level representation, 38
- MRO files, 19
- music transcription, 13
- notation bar sequence, 25
- note events, 17
- off-line, 9
- on-line, 9
- optical music recognition, 18
- optimum predecessor position matrix, 29
- parent-level
  - work, 44
- partial synchronization, 119
- performance bar sequence, 48
- performance block index sequence, 122
- query, 12, 40
- raster group, 71
- rasterization, 18
- reference manifestation, 59
- rendering, 18
- rule set, 76
- score
  - block sequence, 121
  - following, 9
  - track, 50, 104
  - viewer, 144
- score-performance matching, 9
- scoring, 18
- semantic interpretation, 18
- sequence
  - default bar, 25
  - expected performance block index, 124
  - notation bar, 25
  - performance bar, 48
  - performance block index, 122
  - score block, 121
- source bar, 121
- staff signature, 101
- step
  - decision cost, 134
  - weights, 30
- symbolic, 9
- score data, 17
- synchronization
  - audio-audio, 9
  - cross-domain, 9
  - partial, 12, 119
  - score-audio, 9
  - sheet music-audio, 24
  - symbolic-audio, 9
  - symbolic-symbolic, 9
- target bar, 121
- track
  - audio, 50
  - identification, 50, 103
  - leaf-level, 50
  - score, 50, 104
  - segmentation, 50, 103
  - start browser, 144
- typesetting, 18
- vector shapes, 17
- violations, 76
- Viterbi algorithm, 32
- voice, 71
- waveform data, 17
- work, 44
  - leaf-level, 44, 103
  - parent-level, 44, 103